AMORPH.pro
SMARTUNIFIER

SMARTUNIFIER User Manual

Amorph Systems GmbH

Apr 28, 2026

ABOUT SMARTUNIFIER


Integrate perfectly your
Production-IT using
SMARTUNIFIER
ADVANCED IT-INTEGRATION PLATFORM
AMORPH.pro
SMARTUNIFIER

CHAPTER

ONE

ABOUT SMARTUNIFIER

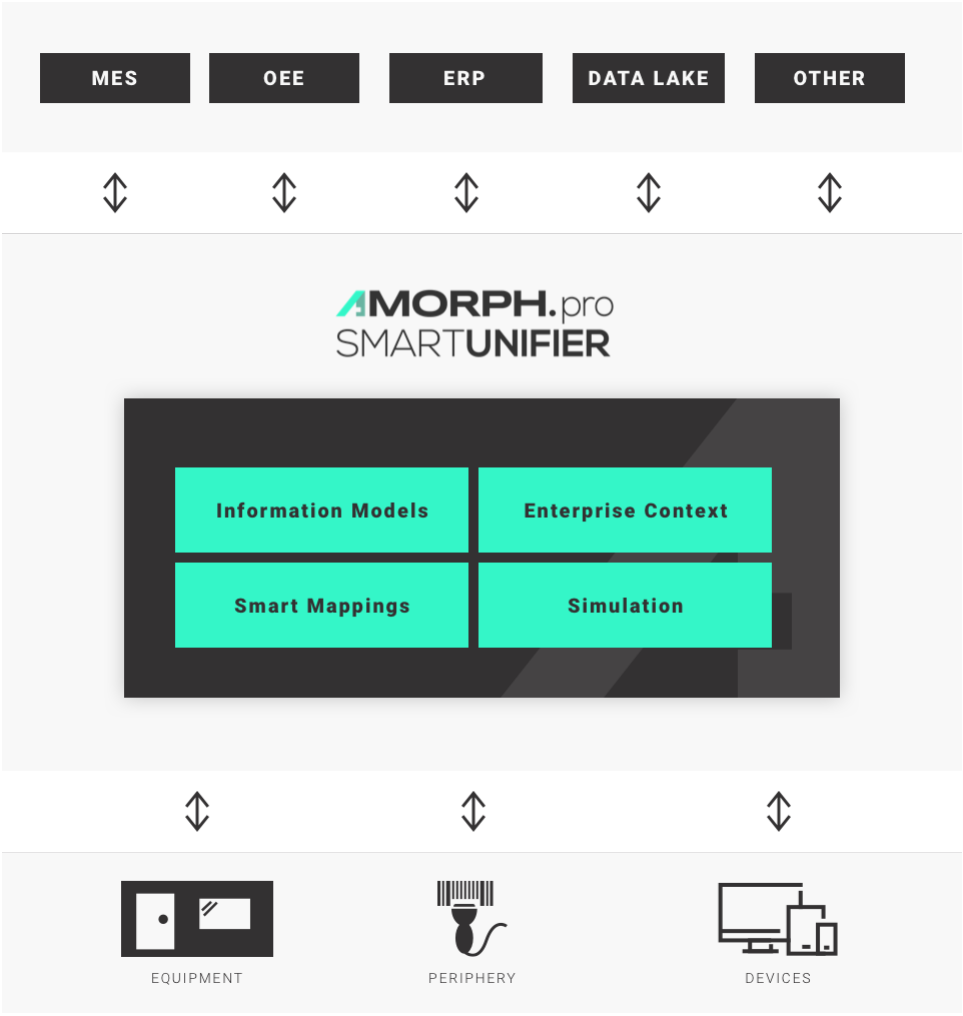
You are new to SMART**UNIFIER**?

- Learn about the *SMARTUNIFIER* connectivity platform
- Learn about the *connectivity use cases* you can address with SMART**UNIFIER**
- *Get started* with your integration

What is SMARTUNIFIER

SMART**UNIFIER** represents a powerful but very easy to use decentralized industrial connectivity platform for interconnecting all industrial devices and IT systems including equipment, peripheral devices, sensors/actors, MES, ERP as well as cloud-based IT systems.

SMART**UNIFIER** is the tool of choice for transforming data into real value and for providing seamless IT interconnectivity within production facilities.



What does SMARTUNIFIER do

- SMART**UNIFIER** provides an easy way to collect data from any *Data Source* and is able to transmit this data to any *Data Target*.
- Data Sources and Data Targets (commonly referred to as Communication Partners) in this respect may be any piece of equipment, device or IT system, communicating typically via cable or Wi-Fi and using a specific protocol like e.g., OPC-UA, file-based, database, message bus.
- With SMART**UNIFIER** several Communication Partners can be connected simultaneously.
- With SMART**UNIFIER** it is possible to communicate unidirectional or bidirectional to each Communication Partner. i.e., messages and events can be sent and received at the same time.
- SMART**UNIFIER** can translate and transform data to any format and protocol that is required by a certain Data Target. This includes different pre-configured protocols and formats, e.g., OPC-UA, file-based, database, message bus, web services and many direct PLC connections. In case a certain protocol or format is currently not available it can be easily added to SMART**UNIFIER**.
- By applying so called *Information Models*, SMART**UNIFIER** enables the same view to data regardless of the protocol or format being used to physically connect an equipment, de-

vice or IT system.

- A big advantage of SMART**UNIFIER** is, that in many cases there is no need for coding when providing interfaces between different Communication Partners – providing a new interface is just drag and drop of data objects between data source(s) and destination(s).

Important Use Cases with SMARTUNIFIER

SMART**UNIFIER** enables an easy and very efficient realization of many use cases that are crucial for gaining Industry 4.0 Excellence.

In the following subchapters some of the most important SMART**UNIFIER** Use Cases are described. These give a comprehensive overview of the advanced SMART**UNIFIER** Features.

Anything-To-Anywhere IT Interface

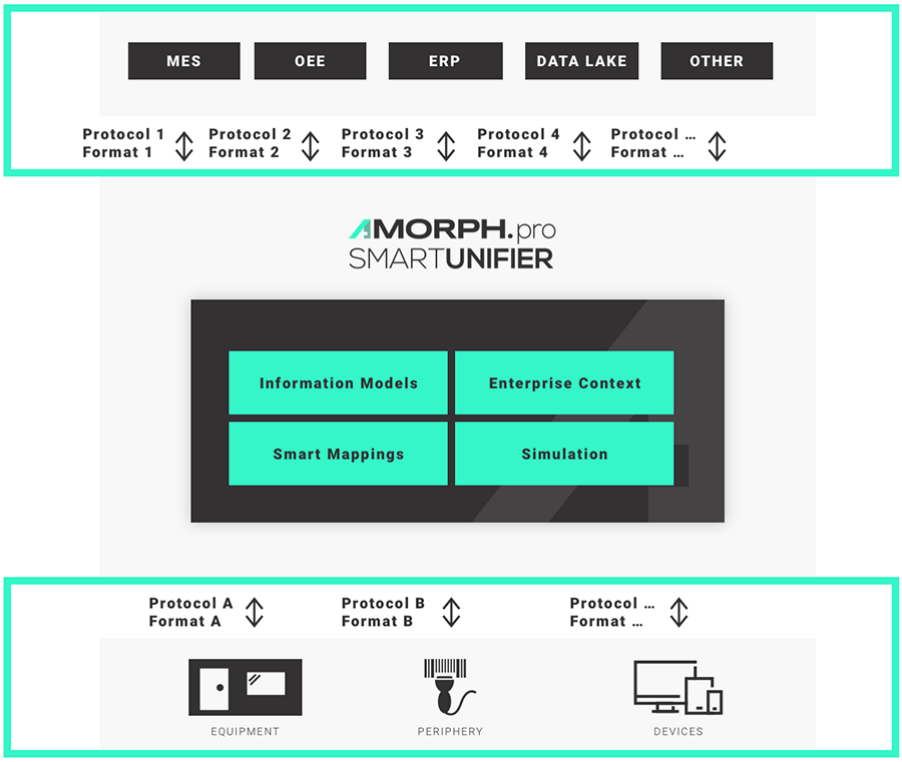
Easy, fast and flexible bi-directional interconnection of multiple IT systems and equipment within a production facility.

Interconnecting heterogeneous shop floor equipment and devices with IT systems and interconnecting different IT systems with each other is a central requirement for a successful transition to modern Industry 4.0 IT landscapes.

SMART**UNIFIER** offers the unique capability to easily interconnect equipment and devices by allowing

- any number of parallel high-speed *Communication Channels* between equipment, devices and IT systems
- high-speed translation between different communication protocols and formats by applying configurable and reusable *Information Models* and *Smart Mappings*
- flexible integration of equipment periphery
- easy integration of enterprise-specific information (e.g., equipment -location/-name/-type/-capabilities) via configurable Enterprise Context
- riskless simulation of interfaces and communication scenarios

Results from renowned reference customers have shown that average equipment integration efforts and **cost can be reduced by up to 90%** using the SMART**UNIFIER** and its advanced technologies to perform powerful IT integration by configuration instead of tedious interface programming.



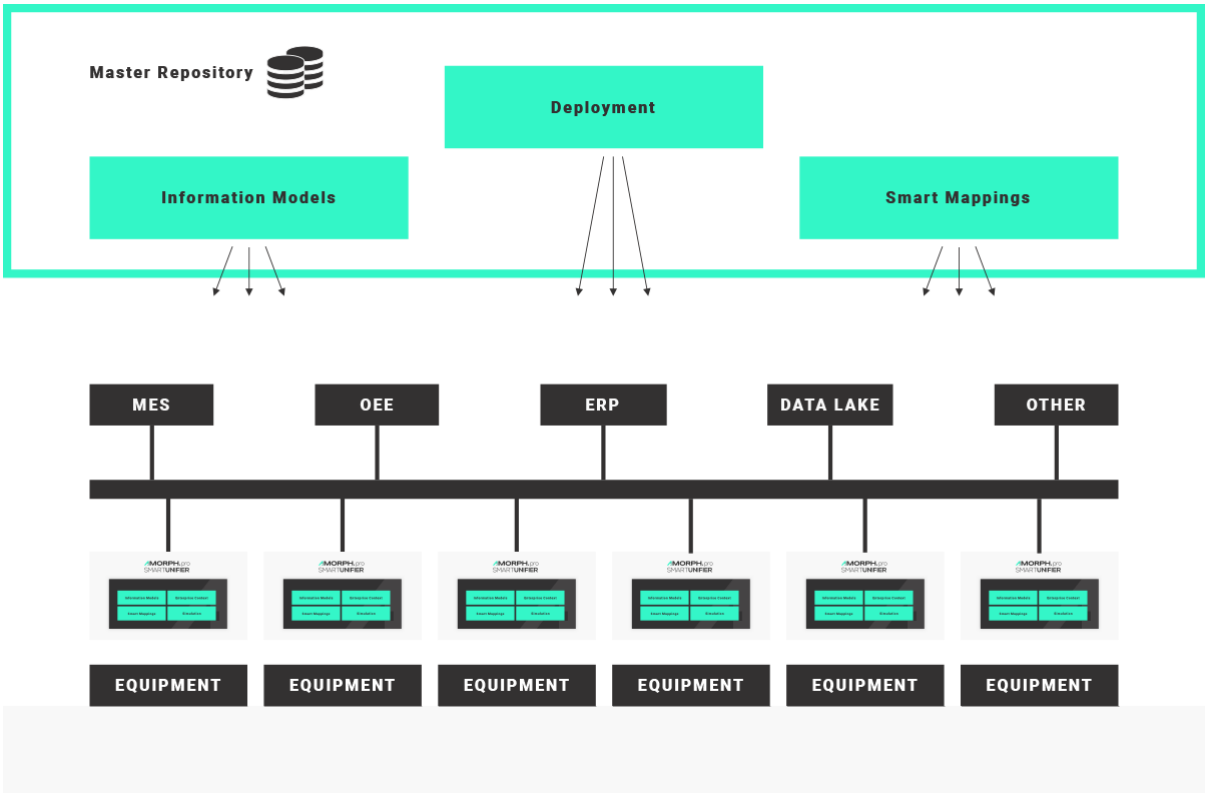
Reusable Interfaces and Interface Models

Reuse interface configurations multiple times with minimum effort.

When running an IT network with a higher number of installed SMART**UNIFIER** *Instances*, all previously created interface configurations (Information Models and Smart Mappings) can be reused easily and shared across the whole installation. This way similar equipment types are integrated using the same connection and translation logic.

Changes and updates of interface configurations can be deployed from a centrally accessible Master Repository, eliminating the need to touch and update each equipment or device individually

Summarized, SMART**UNIFIER** allows a highly comfortable and effective management of very small to very large IT communication environments, creating minimum overhead and letting you reach your main goal: Excellent Manufacturing with a full Industry 4.0 IT infrastructure.



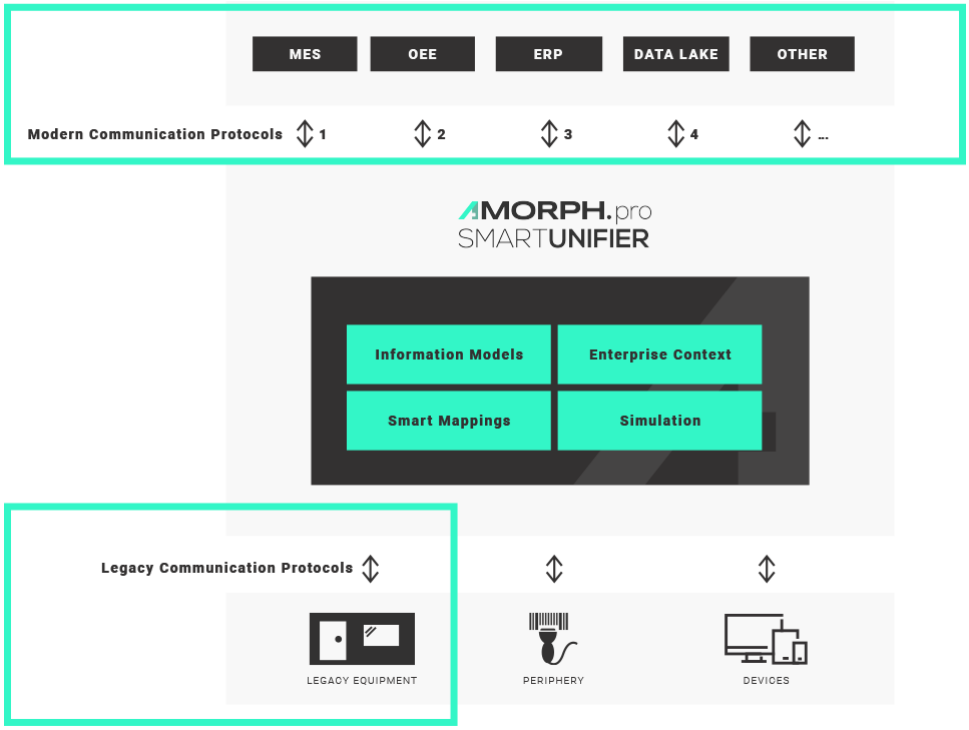
Integrate Legacy Equipment

Fast adaptation of legacy communication protocols and formats to modern enterprise standards.

By applying SMART**UNIFIER** configurable protocol translation (Smart Mappings), modern communication standards like OPC-UA or XML over message bus are fully supported.

SMART**UNIFIER** allows a smooth migration from existing communication protocols and formats (e.g., between existing equipment and MES) to new Industry 4.0 standards.

This unique capability of SMART**UNIFIER** is realized by simply using existing communication channels simultaneously with newly introduced channels. When finishing the migration, the old channels can be switched off without any risk.

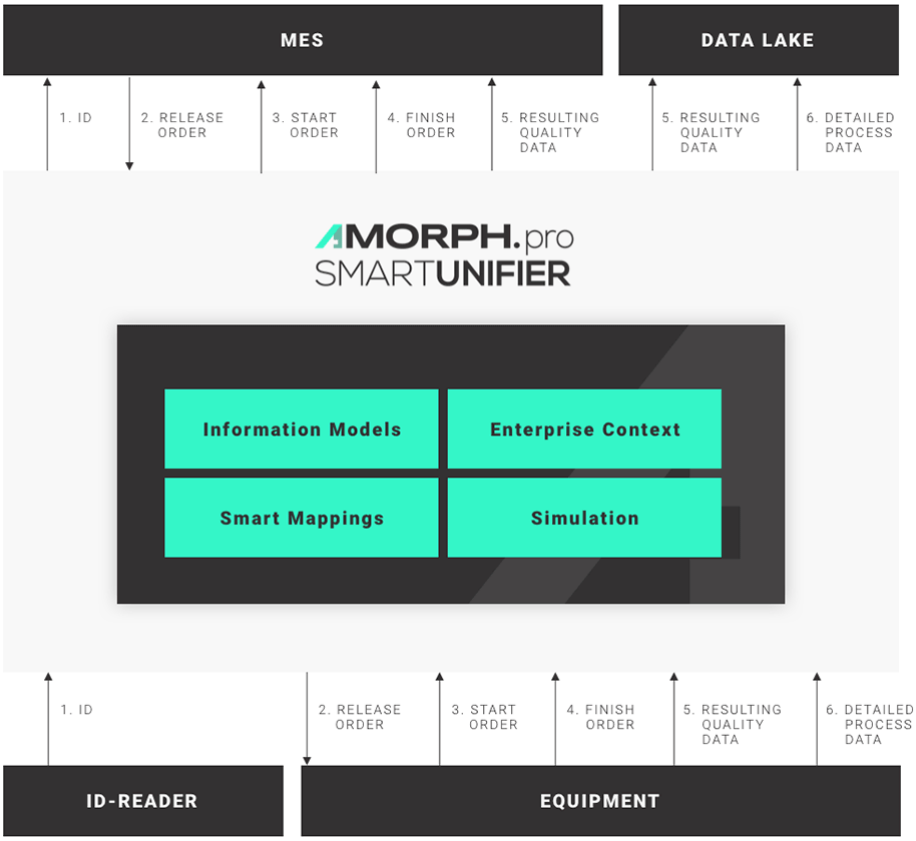


Implement Fab Communication Scenario

Easily implement complete fab communication sequences that cover multiple steps.

With SMART**UNIFIER** it is not only possible to give access to simple equipment or device data and to provide „some data to MES and Cloud“, but also with SMART**UNIFIER** complete communication scenarios between equipment to upper-level IT systems can be easily implemented.

The communication scenarios can cover all steps from identification, validation, order start as well as sending results and process data from equipment to MES or Cloud. Of course, it is also possible to provide any parameter data (recipes) from MES or SCADA to equipment.



Provide Base for Remote Maintenance and Health Monitoring

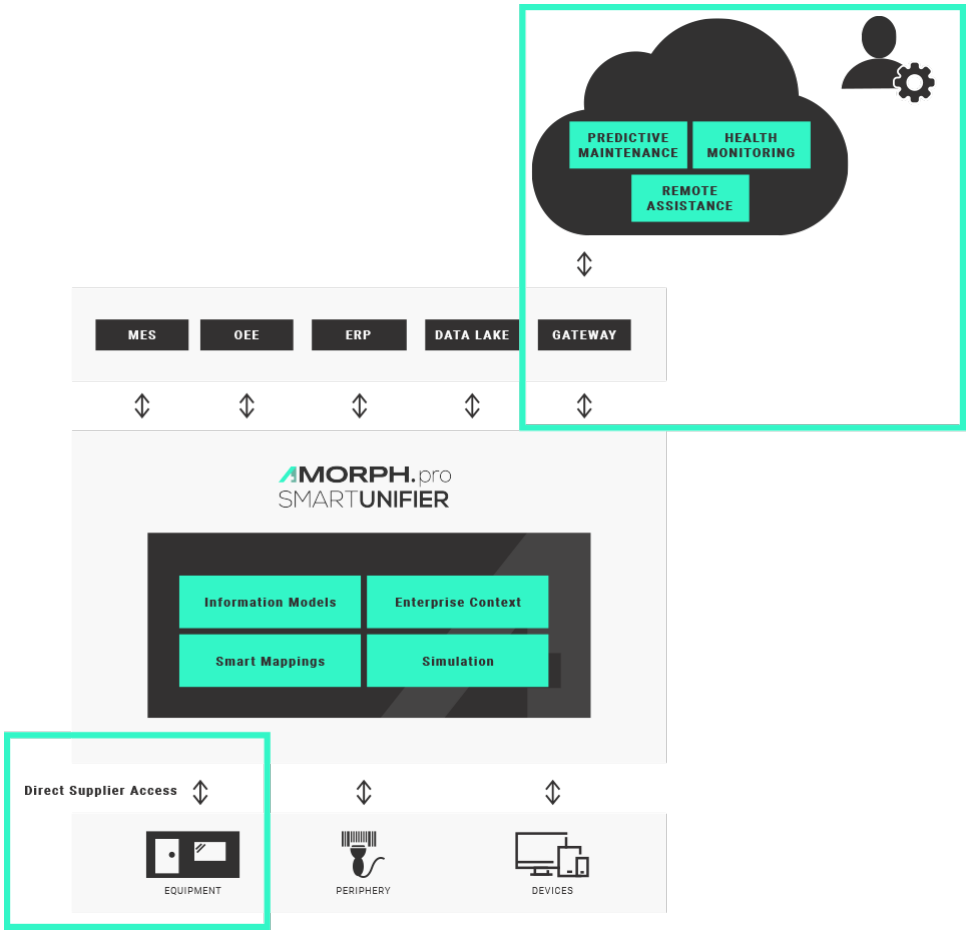
Establish new services and business models by giving secured multi-channel access to equipment and device data in real-time.

Production equipment can be integrated with SMART**UNIFIER** to provide direct access for equipment suppliers or maintenance service providers to relevant equipment data (e.g., equipment status, equipment key parameters) via an equipment supplier's cloud infrastructure.

This way, new innovative business models for equipment suppliers are supported by building the base for "Production as a Service" offerings and remote predictive maintenance.

Also, further advanced business use cases with SMART**UNIFIER** are possible, e.i., by implementing real-time equipment monitoring capabilities in a cloud environment.

Another SMART**UNIFIER** use case is to give Remote Assistance to equipment suppliers to achieve production optimization and to ensure the most efficient usage of equipment resources for customers.



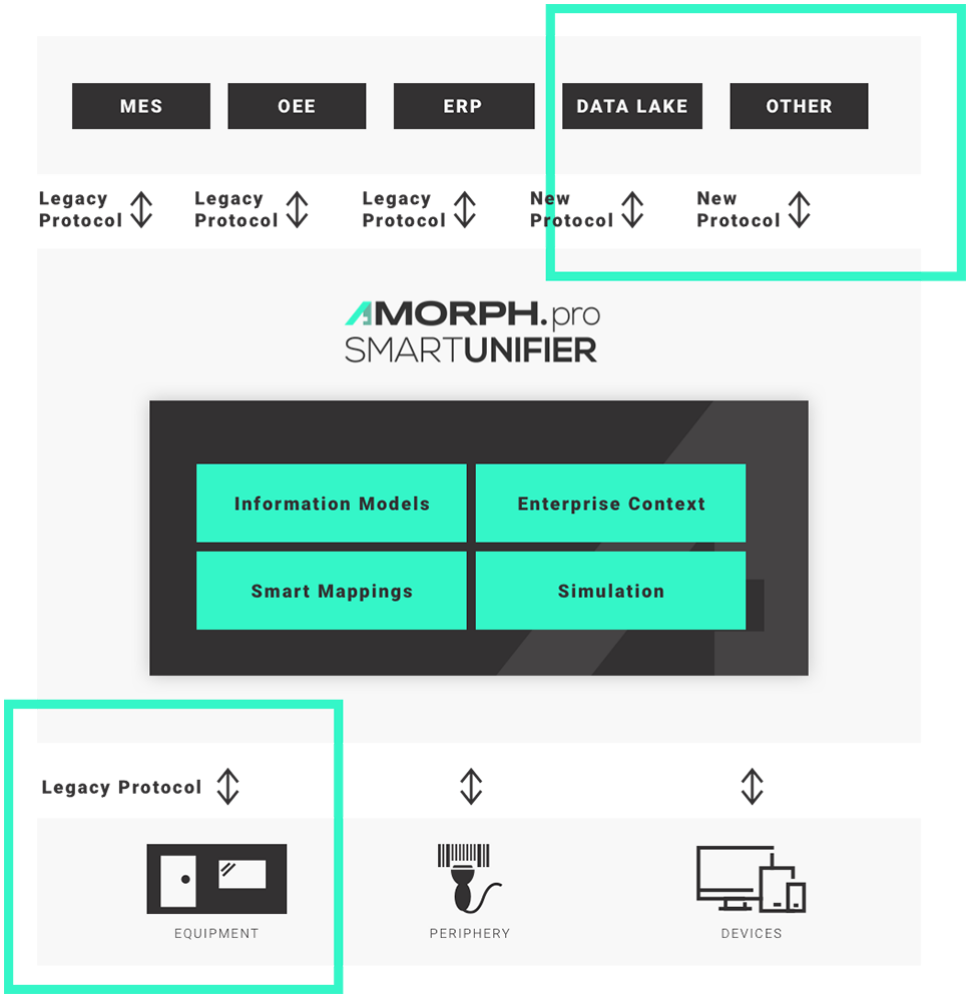
Migrate to Industry 4.0

Migrate step by step to modern communication standards and apply enterprise-wide semantics to data.

A key feature of SMART**UNIFIER** is to open an easy way to integrate new IT systems using modern communication protocols. This is realized by simply adding additional communication channels to the existing legacy channels.

Another feature of SMART**UNIFIER** in this respect is, that all existing IT systems with their legacy protocols and formats can still be operated in parallel with the newly established IT systems (e.g., Data Lake, Advanced Analytics, Cloud).

This way, it is possible to step by step introduce modern communication standards and incrementally migrate to a state-of-the-art Industry 4.0 IT architecture, but still keep the existing IT infrastructure fully operable.



Allow Unlimited Scalability

Rely on unlimited scalability from single equipment and devices to whole facilities.

SMART**UNIFIER** is the first industrial connectivity platform that allows nearly unlimited virtually scalability in terms of number of connected equipment and devices. The SMART**UNIFIER** platform can be applied for integrating one single equipment or device, but with SMART**UNIFIER** hundreds or even thousands of equipment and devices within whole facilities can be integrated to upper-level systems or into the Cloud.

This is because SMART**UNIFIER** is not a traditional middleware having a central limiting message bus. Nor does SMART**UNIFIER** contain any central performance and latency limiting database for providing its communication features.

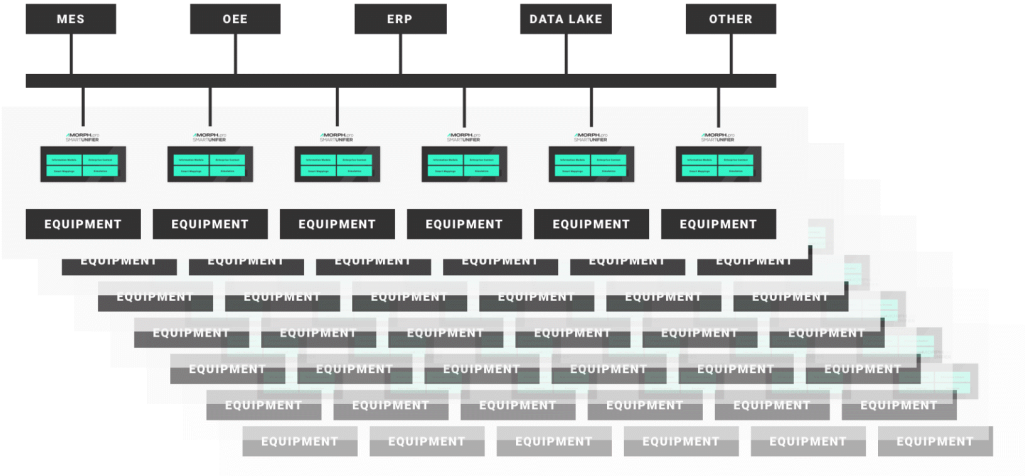
SMART**UNIFIER** works as a distributed environment. Using advanced technologies of distributed computing is the key for enormous scalability.

In a large installation a high number of SMART**UNIFIER** Instances, each with low software footprint, provide the required communication capabilities. These single instances can be deployed to any location within an enterprise IT network – on a server, on an equipment PC, within the Cloud.

Nevertheless, the configuration of all SMART**UNIFIER** Instances can be managed centrally:

- central configuration of Information Models and Smart Mappings
- central Operations Monitoring of installed SMART**UNIFIER** Instances.

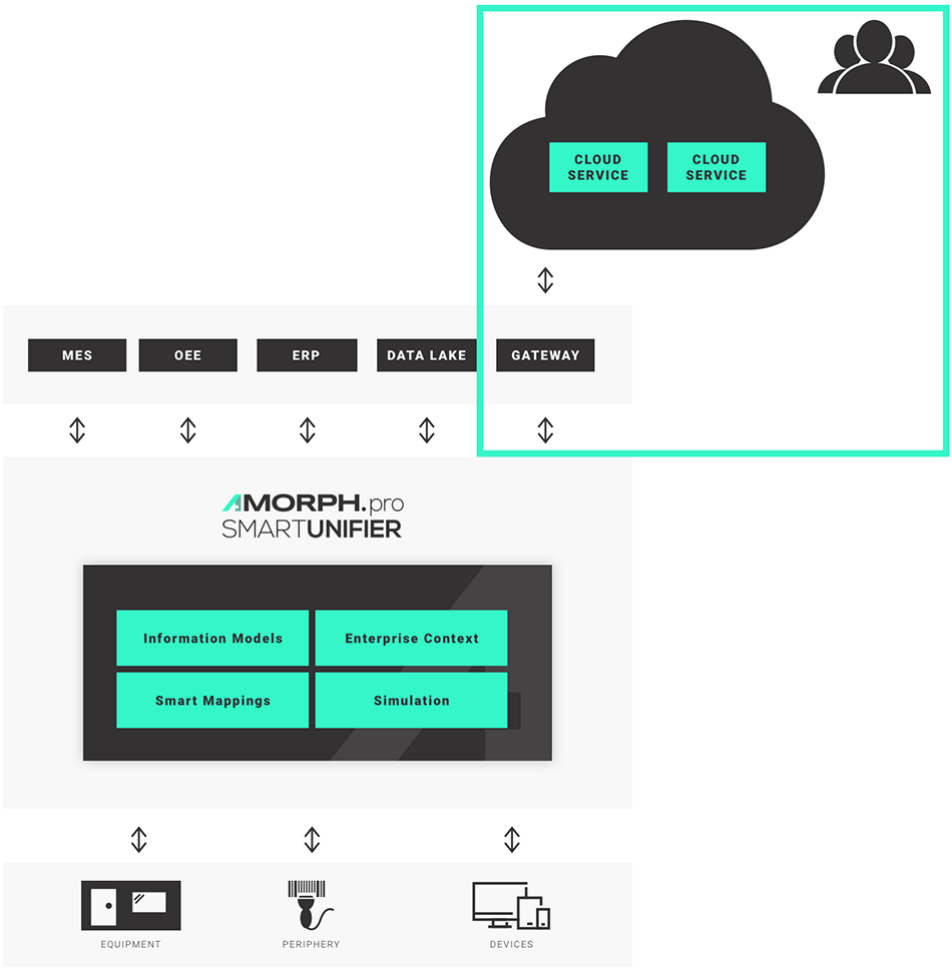
Thus, SMART**UNIFIER** is an essential piece of Industry 4.0 for any manufacturing enterprise – allowing fab-wide and enterprise-wide management of production communication and IT integration infrastructure.



Enable Internet of Things

Out-of-the-box connections between equipment, devices and other IT systems to Cloud infrastructures.

By acting as a translator between equipment and any IOT device precise and secured access of data consumers is possible. The easy connection to any Cloud based infrastructure is also possible (e.g., AWS, Azure).



Getting Started

Integration of industrial equipment, periphery and devices with IT systems can become a quite complex task. SMART**UNIFIER** delivers a standard way of implementing such integration scenarios. We recommend following the procedures described below.

Step	Action (Phases)	Description
1	Preparation	Collect the requirements of systems that are going to be integrated: • Find out what protocols are in use. • Identify the data structures. • Identify the overall communication scenarios (creation of sequence diagrams might be useful). Test and validate the communication of the systems that are going to be integrated (e.g., testing basic connections to the systems using tools like MQTT Explorer). Define a plan on how to conduct the testing before the rollout.
2	Deployment of Manager	The number of SMARTUNIFIER Manager installations depends on the scope of the integration. If several plants are involved, it is recommended to have one installation per plant. **Note:** Reusable configuration components such as Information Models can be shared across multiple SMARTUNIFIER Manager installations (Backup and Restore).
3	Configuration of the Communication Instance	For each Communication Instance: • Create Information Models based on the data structure of the system that is going to be integrated. • Create and configure the Communication Channels that are needed to connect to the systems. • Create the Mapping between the Information Models of the systems based on the previously defined communication scenario. • Create the Device Type that acts as a template for the Communication Instance. • Create and configure the Communication Instance.
4	Deployment of the Communication Instance	For each Communication Instance: • Plan the deployment of the Communication Instance. • Determine the proper deployment type for the Communication Instance (on-premises, edge, or cloud deployment). • Deploy the Communication Instance accordingly.
5	Testing	Test the communication according to the previously defined test plan.
6	Rollout	Rollout and scaling.

CHAPTER

TWO

INSTALLATION

Overview

Installing SMART**UNIFIER** on a host is the first step in realizing your connectivity scenario. There are three ways you can install and operate SMART**UNIFIER**:

- Install the SMART**UNIFIER** installation package on your operating system
  - See the *system requirements* prior to the installation
  - Amorph Systems supports using SMART**UNIFIER** on the following operating systems:
    * *Windows*
    * *Linux*
    * *macOS*
- Install SMART**UNIFIER** in an containerized environments using *Docker*

SMARTUNIFIER Setup

Before using SMART**UNIFIER** Manager it is recommended to have a look over the following setup options:

- *Product Information* and *Activation*
- *Credential Management*
- *Enabling HTTPS*
- *Setting up an external version control*
- *Connecting to a remote database*

Planning the Installation

In production it is typically recommended to host SMART**UNIFIER** Manager and the Communication Instances on separate servers.

In a test environment, the Manager and the Communication Instances can run on the same hardware.

The SMART**UNIFIER** Manager and the Communication Instances can be run on dedicated hardware as well as in a virtualized environment. CPU and memory requirements are estimated

based on modern hardware (e.g. Intel Xeon Coffe Lake / Core i5-7xxx or AMD Epyc / Ryzen 5 3xxx or greater). In a virtualized environment, dedicated resources are highly recommended.

Minimum System Requirements (Manager)

- **Computer and Processor**: 4 cores
- **Memory**: 1GB free memory
- **Storage (SSD)**: 10GB free space
- **Display PC (Engineering, Dashboard)**: 1920x1080 (Full HD)
- **Mobile Devices (Dashboard)**: Latest version of Apple iPadOS, Apple iOS, Android
- **Operating System**: Latest version of Windows, Windows Server, Linux, MacOS (Not recommended for production) - For an optimal user experience always use the latest version of the operating system
- **Browser**: Latest version of Chrome, Microsoft Edge or Firefox

Minimum System Requirements (1 Communication Instance)

- **Computer and Processor**: 4 cores
- **Memory**: 256MB free memory
- **Storage (SSD)**: 5GB free space
- **Operating System**: Latest version of Windows, Windows Server or Linux

Production Deployment Example

Multiple SMART**UNIFIER** Instances can be operated on one server. The number of Instances for each server depends on the overall scenario.

For a deplyoment of ca. 30-40 Communication Instances, the servers minimum requirements are:

- **Computer and Processor**: 8 cores
- **Memory**: 16GB free memory
- **Storage (SSD)**: 150GB free space
- **Operating System**: Latest version of Windows Server

For high-end use cases that require high data volumes, low latency and high amount of data pre-processing, it might be that additional computing resources are required (e.g. deploy one single high-performance Communication Instance on one dedicated computing device).

> **Note**
>
> Communication Instances store log files on the host system therefore sufficient storage needs to be provided.

Test Environment Deployment Example

In a test environment the Manager and the Communication Instances can run on the same machine. Hardware configuration for running 20 communication instances as well as the Manager:

- **Computer and Processor**: 8 cores
- **Memory**: 16GB free memory
- **Storage (SSD)**: 100GB free space
- **Operating System**: Latest version of Windows Server

Windows

SMART**UNIFIER** be delivered in two formats: as an **executable (.exe)** or as a **ZIP** archive.

Install SMARTUNIFIER Manager (Archived Package)

Follow the steps below to install SMART**UNIFIER** Manager:

- Move the SMART**UNIFIER** installation package to a suitable location. Make sure the path to the directory does not include any white spaces!
- Extract the **.zip**-archive.
- Execute the **UnifierManager.bat** script. Afterwards the SMART**UNIFIER** Manager Console appears on the screen.
- Enter your **master password**. When starting SMART**UNIFIER** for the first time go to chapter: *Master Password and Administrator Account*.

After successfully starting up the SMART**UNIFIER** Manager, it can be accessed by opening an Internet Browser (e.g., Chrome or Firefox) and navigating to http://localhost:9000. Use the administrator credentials to login.

> **Note**
>
> The console is for information purposes only. It can be moved to any suitable location on your screen or it can be hidden. Nevertheless, do not close it, because the related processes will also be terminated.

Install SMARTUNIFIER Manager as a Service

Apache Procrun

SMART**UNIFIER** includes **Apache Procrun**, a Windows tool that facilitates the installation and execution of Java applications as services. It simplifies the process by integrating the application with the Windows Service Control Manager.

Follow the steps below to install and operate SMART**UNIFIER** Manager as a Service under Windows:

- Move the SMART**UNIFIER** installation package to a suitable location
- Ensure that the directory path does not contain any white spaces
- Extract the **.zip**-archive

- Open a terminal window with *Administrator privileges* within the installation package
- Execute the following commands in the terminal window to:

Listing 1: Install

```
UnifierManagerService.bat install
```

Listing 2: Start

```
UnifierManagerService.bat start
```

Listing 3: Stop

```
UnifierManagerService.bat stop
```

Listing 4: Uninstall

```
UnifierManagerService.bat uninstall
```

NSSM

> **Hint**
>
> SMART**UNIFIER** does not offer official support for NSSM, unlike Apache Procrun. If you choose to use NSSM, you will need to download the NSSM binary separately.

Follow the steps below to install and operate SMART**UNIFIER** Manager as a Service under Windows using **NSSM**:

- Move the SMART**UNIFIER** installation package to a suitable location
- Ensure that the directory path does not contain any white spaces
- Download the latest version of NSSM to a suitable location (Last tested with version 2.24)
- Go to **win64** and copy the **nssm.exe** in the installation package
- Create the **UnifierManagerService.bat** file in the installation package

Listing 5: UnifierManagerService.bat

```
@echo off
cd %~dp0
set JAVA_HOME=%~dp0\jre
set JAVA=%JAVA_HOME%\bin\java.exe
set MANAGER=%~dp0\bin\adaptermanagerweb.bat
set JAVA_OPTS=-Dunifier.administrator.credentials.file="%~dp0/conf/
↪credentials.properties"

del RUNNING_PID
"%MANAGER%"
```

- Open a terminal window with *Administrator privileges* within the installation package

- Execute the following commands in the terminal window to:

Listing 6: Install

```
nssm install SmartUnifierManager "UnifierManagerService.bat"
```

Listing 7: Start

```
nssm start SmartUnifierManager
```

Listing 8: Stop

```
nssm stop SmartUnifierManager
```

Listing 9: Uninstall

```
nssm remove SmartUnifierManager
```

Optional

Listing 10: Set SERVICE_AUTO_START

```
nssm set SmartUnifierManager Start SERVICE_AUTO_START
```

Linux

Follow the steps below to install SMART**UNIFIER** Manager:

- Move the installation package to a suitable location. Make sure the path to the directory does not include any white spaces!
- Extract the **.tar.gz**-archive.

```
tar -xvzf SmartUnifierManager-linux-x64.tar.gz
```

- Start the Manager by opening up a terminal and executing the following commands:

```
chmod +x UnifierManager.sh
```

```
./UnifierManager.sh
```

> **Note**
>
> Execute **StartUnifierManagerInBackground.sh** when the process should run in background. To stop the process execute **StopUnifierManager.sh**.

```
./StartUnifierManagerInBackground.sh
```

```
./StopUnifierManager.sh
```

- Enter your **master password**. When starting SMART**UNIFIER** for the first time go to chapter: *Master Password and Administrator Account*.

After successfully starting the SMART**UNIFIER** Manager, it can be accessed by opening an Internet Browser (e.g., Chrome or Firefox) and navigating to http://localhost:9000.

Note

The console is for information purposes only. It can be moved to any suitable location on your screen or it can be hidden. Nevertheless, do not close it, because the related processes will also be terminated.

Mac OS

Follow the steps below to install SMART**UNIFIER** Manager:

- Move the installation package to a suitable location. Make sure the path to the directory does not include any white spaces!
- Extract the **.tar**-archive.
- Start the Manager by opening up a terminal and executing the following commands:

```
chmod +x UnifierManager.sh
```

```
./UnifierManager.sh
```

Note

If you get the warning "java cannot be opened because the developer cannot be verified" - go to `System Preferences... > Security & Privacy` and click on `Allow Anyway`.

- Enter your **master password**. When starting SMART**UNIFIER** for the first time go to chapter: *Master Password and Administrator Account*.

After successfully starting up the SMART**UNIFIER** Manager, the SMART**UNIFIER** Manager can be accessed by opening an Internet Browser (e.g., Safari, Chrome or Firefox) and navigating to http://localhost:9000.

Note

The console is for information purposes only. It can be moved to any suitable location on your screen or it can be hidden. Nevertheless, do not close it, because the related processes will also be terminated.

Docker

Requirements

The following example shows how to set up SMART**UNIFIER** using Docker Volumes mount to local paths on the machine.

1. Create the following directories:

conf
: Manager configuration files, keystore and database

```
mkdir -p /opt/amorph/smartunifier/manager/conf
```

repository
Storing compiled artifacts

```
mkdir -p /opt/amorph/smartunifier/manager/repository
```

versioning (optional)
Storing of component sources, not required when using an external git server like gitea.

```
mkdir -p /opt/amorph/smartunifier/manager/versioning
```

logs (optional)
Storing of logs files from the manager

```
mkdir -p /opt/amorph/smartunifier/manager/log
```

2. Copy the **conf** and the **repository** folder from the SMART**UNIFIER** installation package into the newly created corresponding volumes:

```
cp -r conf/* /opt/amorph/smartunifier/manager/conf
cp -r repository/* /opt/amorph/smartunifier/manager/repository
```

Note

Edit the **application.conf** in **/opt/amorph/smartunifier/manager/conf** and remove the lines 'javaHome = "jre"'

```
nano /opt/amorph/smartunifier/manager/conf/application.conf
```

3. Create Docker Volumes mounted to the directories just created:

```
docker volume create --driver local \
 --opt type=bind \
 --opt device=/opt/amorph/smartunifier/manager/conf \
 --opt o=bind smartunifier_conf

docker volume create --driver local \
 --opt type=none \
 --opt device=/opt/amorph/smartunifier/manager/repository \
 --opt o=bind smartunifier_repository

docker volume create --driver local \
 --opt type=none \
 --opt device=/opt/amorph/smartunifier/manager/versioning \
 --opt o=bind smartunifier_versioning

docker volume create --driver local \
 --opt type=none \
```

(continues on next page)

(continued from previous page)

```
--opt device=/opt/amorph/smartunifier/manager/log \
--opt o=bind smartunifier_logs
```

Start Up

Go to the SMART**UNIFIER** package and open the **docker** directory with the console.

1. Build docker image

```
docker-compose build
```

2. Start the manager with attached console

```
docker-compose run smartunifier
```

3. Enter Master password and admin user credentials on request

> **Note**
>
> Remove the default user set up in the **credentials.properties** file in order to set the master password and to create a new admin user.

After the setup is done, a credentials file containing the master password can be used to start the manager without having to input the password.

```
docker-compose up -d
```

Product Information and Activation

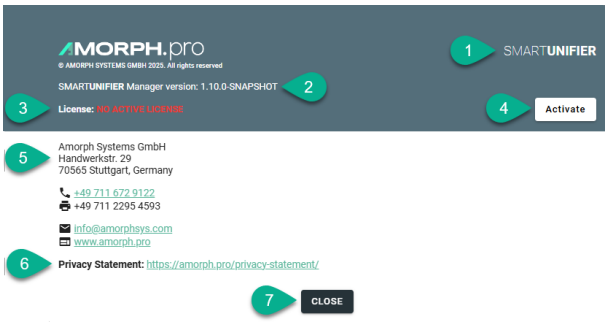
Product Information

To open the product information section click on the **Account** icon **(1)** and select the **About SMARTUNIFIER** section **(2)**.

The **About SMARTUNIFIER** section provides the following information:

1. Product name

2. Manager version
3. License information
4. Activation button
5. Company details
6. Privacy Statement URL



To exit the **About SMARTUNIFIER** section, click on the **Close** button **(7)**.

Product Activation

The SMART**UNIFIER** product requires a license (demo/paid) for activation. The license details are displayed in the **About SMARTUNIFIER** section, as seen above.

The product activation can be done in two ways:

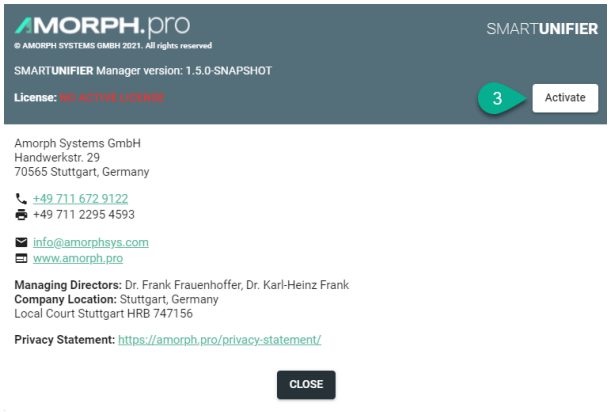
- Online
- Offline

Online Activation

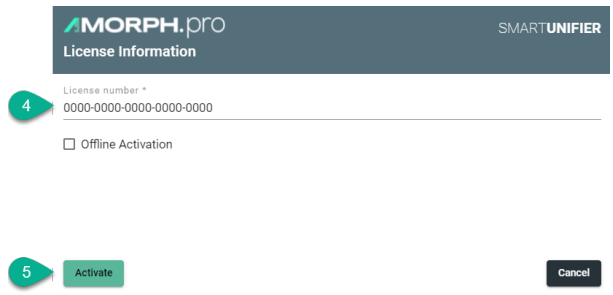
Follow the steps bellow to activate the product online:

- Click on the **Account** icon **(1)** and select the **About SMARTUNIFIER** section **(2)**

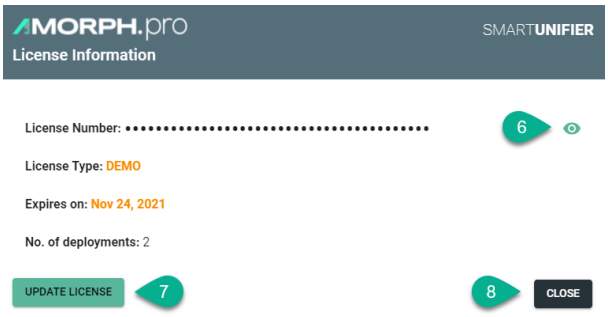
- Click on the **Activate** button **(3)**



- Input the key **license number (4)** and click on the **Activate** button **(5)**



- **The license key is registered, displaying the following details:**
  - License Number, visible by clicking on the **Show** button **(6)**
  - License Type
  - Expiration date
  - Maximum number of available deployments
- Click on the **Update License** button **(7)** to register a new license key or click on the **Close** button **(8)** to exit.

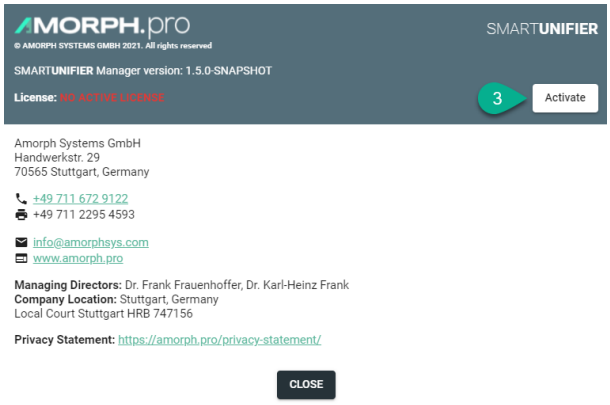


Offline Activation

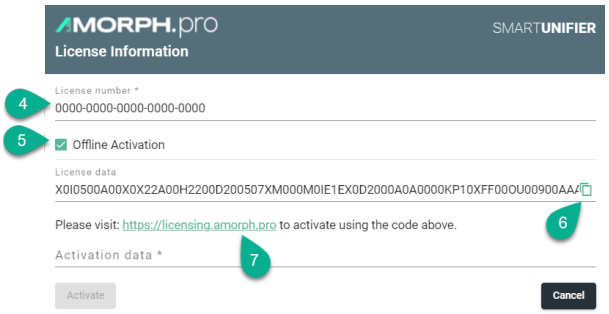
Follow the steps bellow to activate the product offline:

- Click on the **Account** icon **(1)** and select the **About SMARTUNIFIER** section **(2)**

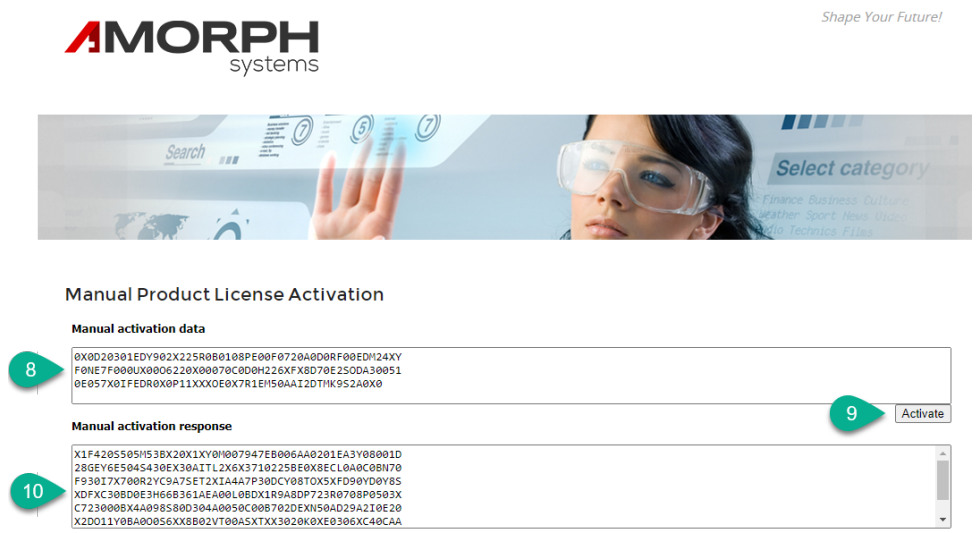
- Click on the **Activate** button **(3)**



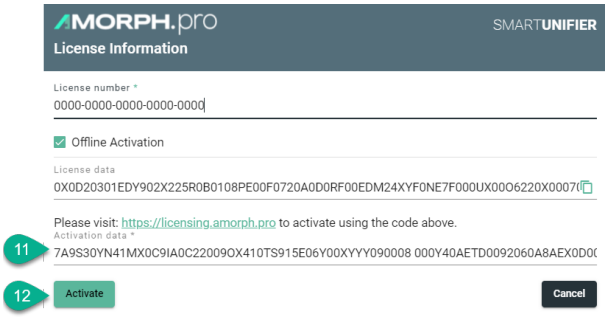
- Input the key **license number (4)** and check the box for **Offline Activation (5)**
- Copy the **License Data (6)** to an external storage unit



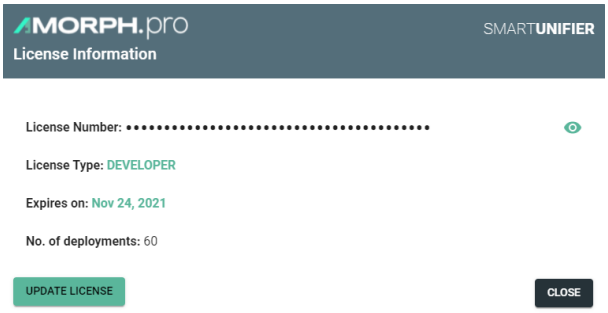
- From a device connected to the internet open the license URL **(7)**
- Paste the **License Data (6)** into the **Manual activation data** field **(8)** and click on the **Activate** button **(9)**



- Copy the **Manual activation response (10)** to an external storage unit and paste it into the **Activation data** field **(11)**



- Click on the **Activate** button **(12)** to finish. The license is registered, as seen bellow.

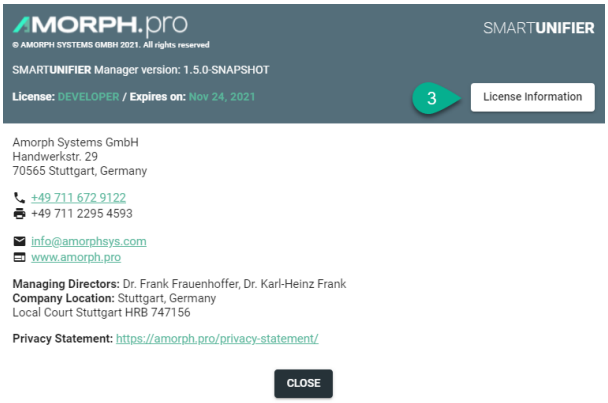


Update License

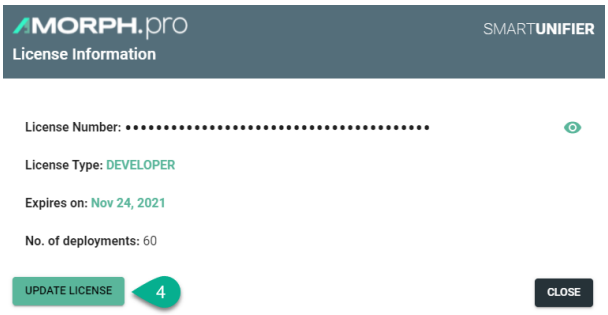
Follow the steps bellow to update the license:

- Click on the **Account** icon **(1)** and select the **About SMARTUNIFIER** section **(2)**

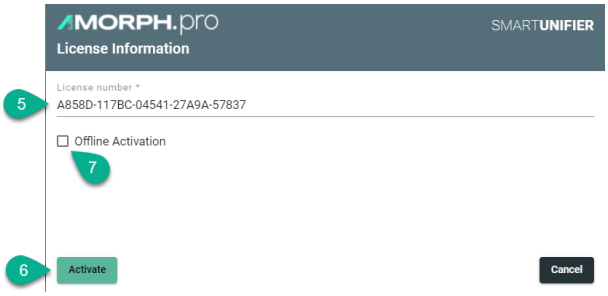
- Click on the **License Information** button **(3)**



- Click on the **Update License** button **(4)**



- Input the key **license number (5)** and continue with *Online Activation* **(6)** or *Offline Activation* **(7)**

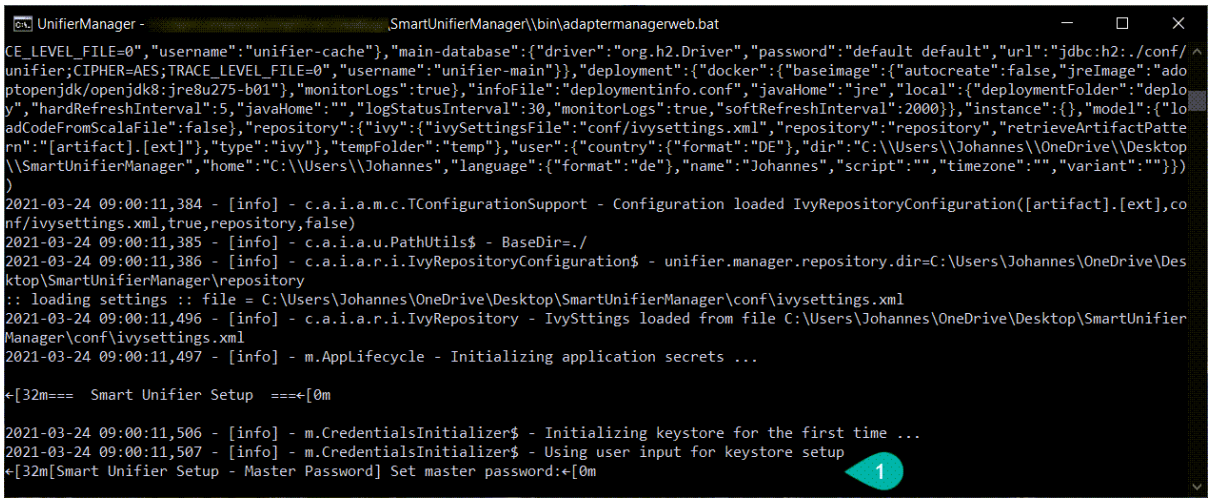


Credentials Management

Master Password and Administrator Account

When starting SMART**UNIFIER** for the first time you will be asked to enter a master password. The master password is needed in order to store credentials securely inside a KeyStore ("**unifier.jceks**") file located on the user's local machine.

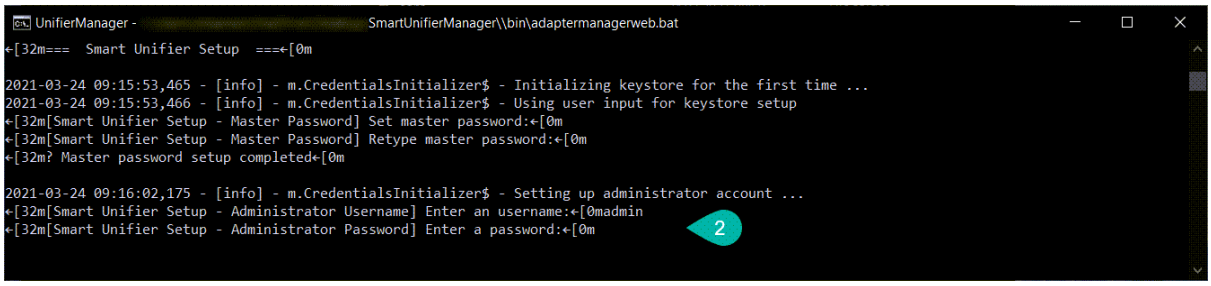
- Enter your master password in the console and re-enter it **(1)**. If the passwords do not match, simply close the console, execute the **UnifierManager.bat** and enter the passwords again.



Warning

If the master password is lost it cannot be recovered!

- Enter the name for the administrator user account and the password **(2)**.



- Open an Internet Browser (e.g., Safari, Chrome or Firefox) and navigating to http://localhost:9000 and login using the administrator account just created.

You can add more users using the SMART**UNIFIER** Manager UI.

Setting default credentials

You can define default credentials to avoid to re-enter the master password on startup.

1. Go to the **SmartUnifierManager** folder
2. Open the file **UnifierManager.bat** for a Windows installation (**UnifierManager.sh** for a installation on Linux/macOS)
3. Add the following line:

```
set JAVA_OPTS=-Dunifier.administrator.credentials.file="%~dp0/conf/
↪credentials.properties"
```

4. Make sure that the file **credentials.properties** exists in the **SmartUnifierManager/conf** folder
   - Set for **unifier.keystore.password** the master password as defined in the chapter *Master Password and Administrator Account*.

Listing 11: credentials.properties file content

```
# Keystore password
unifier.keystore.password=<keystore_password>

# Default Administrator account credentials
unifier.administrator.username=<administrator_username>
unifier.administrator.password=<administrator_password>
```

Enabling HTTPS

Following configuration is required to enable https :

1. Browse to **SmartUnifierManager/conf** folder
2. Open **application.conf** for editing
3. Comment out (using #) following lines

```
play.server.http.port = 9000
play.server.http.address = "0.0.0.0"
```

4. Uncomment following lines and replace path_to_keystore and keystore_password with valid data

```
play.server.http.port=disabled
play.server.https.port=9443

play.server.https.keyStore.path="path_to_keystore"
play.server.https.keyStore.password="keystore_password"
```

5. Save and close

By default, keystore type is JKS. PEM. PKCS12 format is supported. In order to change the keystore type you need to add following configuration: **play.server.https.keyStore.type=PEM**

Generating a keystore is done using the following command:

```
keytool -keysize 2048 -genkey -alias unifier -keyalg RSA -keystore unifier.
↪keystore
```

- keysize 2048 sets the keystore size in bytes. The larger the storage, the more difficult it is to decipher an SSL key. Setting the keystore size to 2048 bytes is sufficient for high-level security.
- genkeypair generates a public key and an associated private key.
- alias unifier sets the alias for the SSL key; use this alias to reference keystore later, when configuring the application.
- keyalg RSA sets the encryption type for storage, which is RSA.
- keystore unifier.keystore, sets the name for the file into which the generated key will be written

Next, you will "fill in a questionnaire". The data you provide is stored in the SSL key.

Once the keystore is created, you can generate a public SSL key. Recall the keystore password and run the following command (the terminal asks you to provide the correct password):

```
keytool -certreq -alias unifier -file unifier_csr.txt -keystore unifier.
↪keystore
```

- certreq generates a public SSL key (which has also the name Certificate Signing Request).-alias unifier sets the alias to refer to the key.
- file unifier_csr.txt creates a unifier_csr.txt file to store the key (this is different from the keystore).
- keystore unifier.keystore sets the key storage file.

You can skip this section if you are going to only test the HTTPS connection. However, if you are going to use the generated SSL key for production, you need to send it to a Certificate Authority.

Copy the SSL key that you can find in the home/johndoe/csr.txt file. **Note** that you must copy the entire contents of the file including the delimiters ----BEGIN NEW CERTIFICATE REQUEST---- and ----END NEW CERTIFICATE REQUEST----. Without the delimiters, your key is not valid.

The SSL provider gives you two certificates in exchange for the key — the root and the intermediary certificates. (These certificates are called primary and secondary.) Add them both into the keystore.

Use the following command to add the intermediary certificate to the keystore:

```
keytool -importcert -alias secondary -keystore unifier.keystore -file <path_
↪to_secondary_certificate>.<ext>
```

- importcert tells the keytool library to import the certificates into storage.
- alias secondary sets the alias for the intermediary certificate.
- keystore unifier.keystore sets the necessary keystore for the certificate.
- file <path_to_intermediary_certificate>.<ext> sets the path to the file with the intermediary certificate.Remember to replace the <path_to_secondary_certificate> with the actual path; and also use the proper file extension instead of <ext>.

Similarly, you can add the root certificate to your storage, in this case you need to use a different command:

```
keytool -importcert -alias unifier -keystore unifier.keystore -trustcacerts -file <path_to_root_certificate>.<ext>
```

External Version Control

Gitea

For the setup, make sure you meet the following prerequisites:

- A local installation of Gitea .
- An user account explicitly for SMART**UNIFIER** - `smartunifier`.
- An access token for this user account.

Once all prerequisites are met continue to authenticate SMART**UNIFIER** to access Gitea:

1. Go to the `application.conf` file that is located in the SMART**UNIFIER** package `SmartUnifierManager-windows-x64\conf`
2. Add the **sourceControl** JSON object inside the **unifiermanager** JSON object:

```
sourceControl {
  gitea = {
  baseUrl="**Enter Url here**"
  accessToken="**Enter access token here**"
  }
}
```

Examples for configuration properties:

Property	Example
baseUrl	`http://localhost/api/v1`
accessToken	`8748ea571d0395434ee1a0a6f46163ba32d8c95e`

Local

Configuration of the local version control:

1. Go to the `application.conf` file that is located in the SMART**UNIFIER** package `SmartUnifierManager-windows-x64\conf`
2. Add the **sourceControl** JSON object inside the **unifiermanager** JSON object:

```
sourceControl {
  localgit {
  repoFolder = "**path to local direcotry**"
  }
}
```

External Database

To connect SMART**UNIFIER** Manager to a remote database follow the steps below:

1. Go to the `application.conf` file that is located in the SMART**UNIFIER** package `SmartUnifierManager-windows-x64\conf`
2. Add the **database** JSON object inside the **unifiermanager** JSON object:

```
database {
  main-database {
      driver = "net.sourceforge.jtds.jdbc.Driver"
      url = "jdbc:jtds:sqlserver://<ip>:<port>;DatabaseName=unifier_db"
      username = "<username>"
      password = "<password>"
  }

  cache-database {
      driver = "net.sourceforge.jtds.jdbc.Driver"
      url = "jdbc:jtds:sqlserver://<ip>:<port>;DatabaseName=cache_db"
      username = "<username>"
      password = "<password>"
  }
}
```

CHAPTER

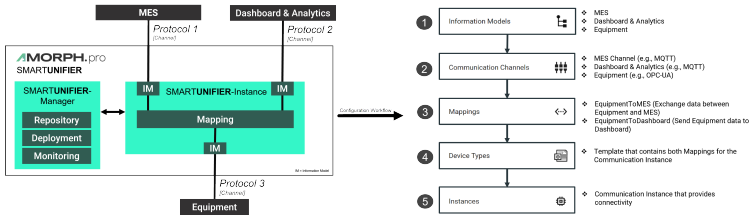
THREE

HOW TO INTEGRATE WITH SMARTUNIFIER

Each integration scenario follows the same workflow, which consists out of 5 steps:

- *Information Models* - describe and visualize communication related data using hierarchical tree structures
- *Communication Channels* - describe and configure the protocols needed for the scenario
- *Mappings* - define when and how to exchange/transform data between Information Models
- *Device Types* - define templates for Instances
- *Instances* - define applications that provide the connectivity

Below you can see an example of integration scenario and the necessary steps to establish connectivity with SMART**UNIFIER**:



Information Models

What are Information Models

In the SMART**UNIFIER**, an Information Model is defined as the data related to communication that a device or IT system can provide. Each device or IT system is represented by an Information Model. An Information Model is composed of elements known as *Node Types*, and these models are structured hierarchically like a tree. This means that elements within an Information Model can include additional elements. This hierarchical structure is essential for accurately modeling the data structure of devices in a way that reflects their real-world complexity.

Before starting the modeling process for the data structure of, for example, equipment or an IT system, it's crucial to have a clear understanding of the overall use case. Generally, each piece of equipment or IT system that needs to be integrated will have its own Information Model. However, if there are multiple pieces of equipment or IT systems of the same type, a single Information Model can be used for all of them.

During the later stages of configuring a SMART**UNIFIER** instance, the Information Model that is included in a Mapping, as well as the Mapping itself, can be applied to a Device Types. This

allows for the creation of multiple instances from a single device type. Therefore, the Information Model serves as a blueprint for the data structure of a device or IT system and needs to be created only once. Understanding the reusability of the Information Model is crucial.

This concept emphasizes that once an Information Model is established, it can be effectively reused across different devices or IT systems of similar types, significantly streamlining the configuration process and enhancing efficiency.

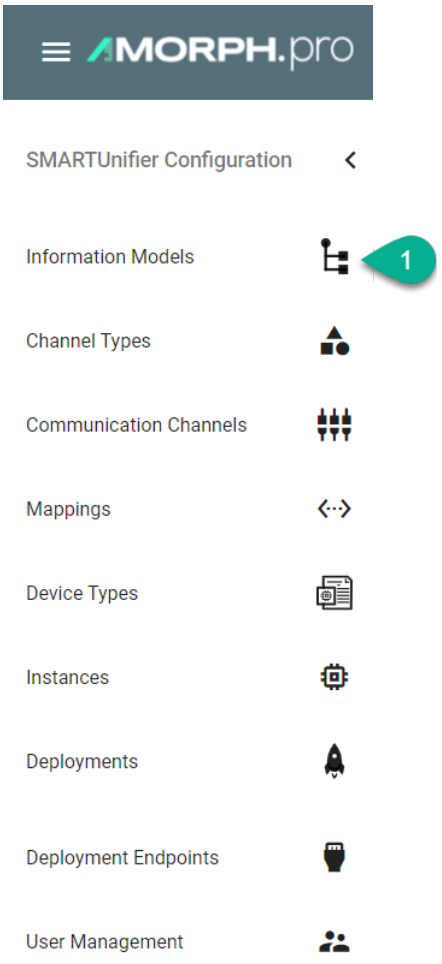
Contextualization

Contextualization describes the process and functionality of connecting and combining correlated data, often across system boundaries. This context data can be stored and used internally within SMART**UNIFIER** by utilizing Information Models to define the data structures, in combination with the *InMemory* Communication Channel. The context data can then be reused and mapped to other systems wherever the data is needed.

How to create a new Information Model

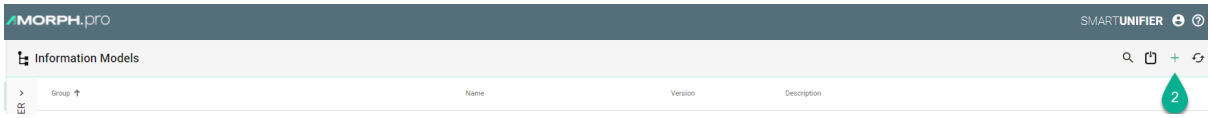
Follow the steps described below to create an Information Model:

- Select the SMART**UNIFIER** Information Model Perspective **(1)**.

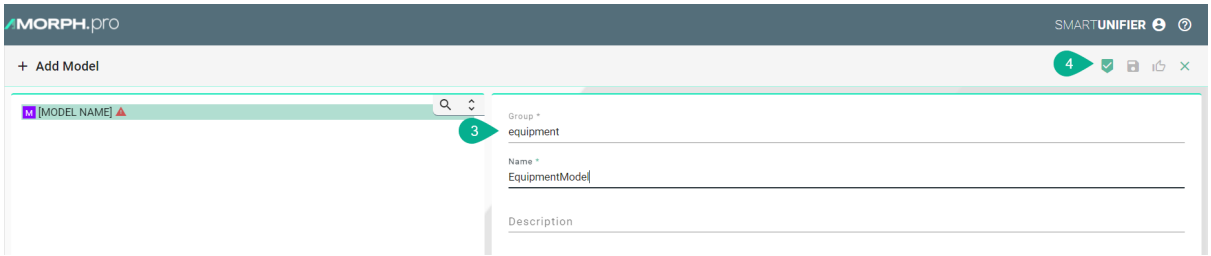


- You are presented with the following screen containing a list view of existing Information Models.

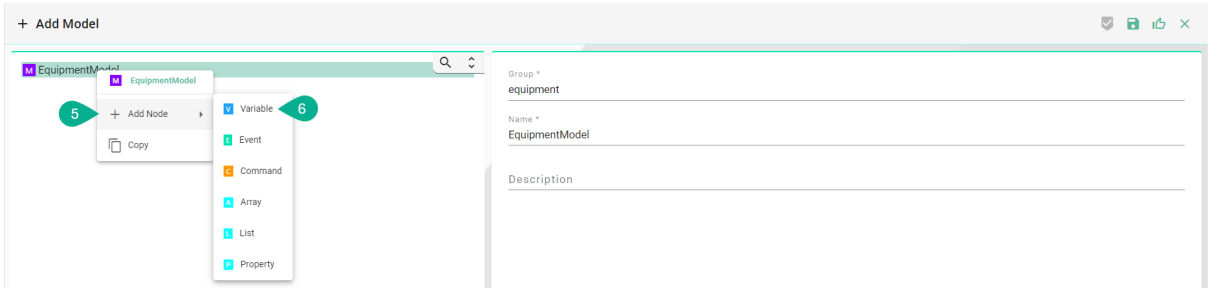
- In order to add a new Information Model, select the “Add Model“ button at the top right corner **(2)**.



- On the following screen provide the following mandatory information: Group and Name **(3)**.
- The “Apply” button at the top right corner is enabled after all mandatory fields are filled in. Click the button to create a new Information Model **(4)**.
- The newly created Information Model is now visible as a node on the left side of the screen.



- After the root model node is created, a new Information Model can be built up using definition types.
- Perform a right click on the root model node and select "Add Node" **(5)**. Select a Definition Type from the dialog **(6)**.



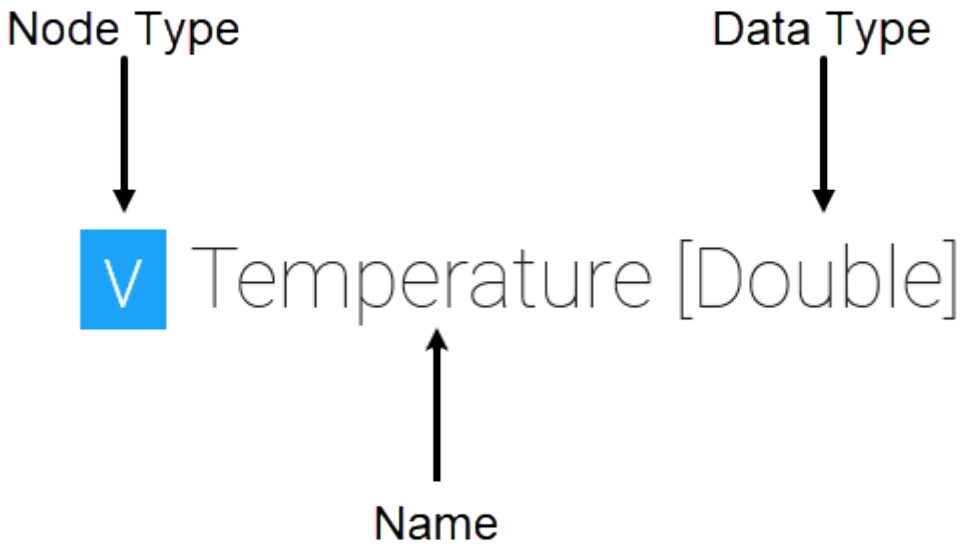
Node Types

Basics

Based on the specific use case and the communication channels involved, it is important to select the appropriate Node Type. Node Types are key components of an Information Model and include various elements such as variables, properties, events, commands, and collections like arrays and lists.

Each Node Type is associated with a Data Type, which specifies whether it is a predefined data type (such as String, Integer, Boolean, etc.) or a custom data type.

Example of a Node of Type Variable:



Naming Restrictions

There are specific restrictions on the naming conventions for Node Types. For instance, *Scala keywords* are not permissible. Moreover, a Node Type name cannot begin with a number, and the underscore character "_" is the sole permitted separator.

If the name of the Node, for example, the name of a Variable, requires a specific naming convention that cannot be used as a Node Type name, it can be customized using the "Field Name" option within the corresponding Communication Channel. Supported Communication Channels with the option to set a custom field name include:

- File Reader (CSV, JSON, XML)
- File Tailer (CSV, JSON, XML)
- File Writer (CSV, JSON, XML)
- MQTT (CSV, JSON, XML)
- SFTP File Writer (CSV, JSON, XML)
- WebSocket Client (CSV, JSON, XML)

Hint

If you need to use a specific naming convention for a Node Type where restrictions apply and the Channel does not support custom field names, please contact the SMART**UNIFIER** support team.

List of Scala Keywords Prohibited as Node Type Names in the Information Model:

Scala keywords					
abstract	else	lazy	override	super	true
case	extends	match	package	this	try
catch	false	new	private	throw	type
class	final	null	protected	trait	val
def	finally	object	return	true	var
do	for	if	sealed	type	while
else	forSome	implicit	super	val	with
yield					

Available Node Types

Variables

What are Variables

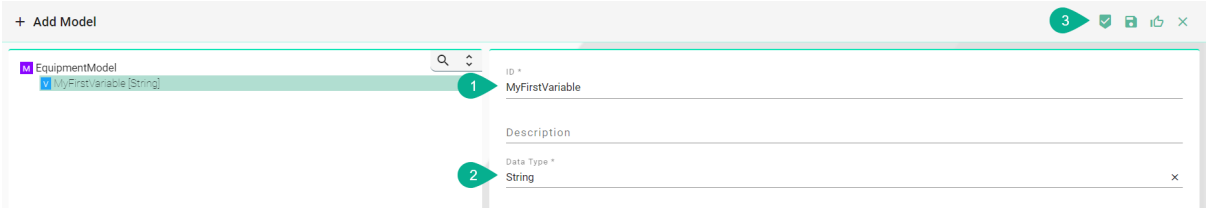
Variables in an Information Model are components that can contain either a specific value or a structure composed of other variables.

Data Types

- For variables intended to store specific values, use the *Simple Data Type*.
- To create structures, use the *Custom Data Type*.

How to create a Variable

- Enter an ID **(1)**
- Enter a Data Type **(2)**
- Click the "Apply" button **(3)**



Properties

What are Properties

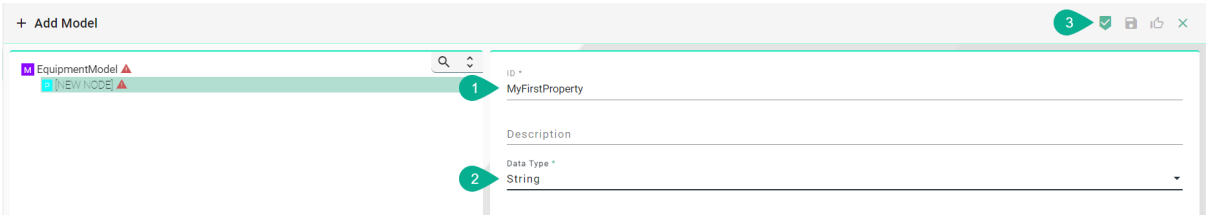
Properties are working similar to *Variables*. Properties can be used for XML attributes when XML-files are subject to be processed by SMART**UNIFIER**, although XML elements are still represented by Variables in the *Information Model*.

Data Types

Property Node Types can be defined using *Simple Data Types*.

How to create a Property

- Enter an ID **(1)**
- Enter a Data Type **(2)**
- Click the "Apply" button **(3)**



Events

What are Events

SMART**UNIFIER** is an event-driven software. In this context an event is an action or occurrence recognized by SMART**UNIFIER**, often originating asynchronously from an external *data source* (e.g., equipment, device), that may be handled by the SMART**UNIFIER**. Computer events can be generated or *triggered* by external IT systems (e.g., received via a *Communication Channel*), by the SMART**UNIFIER** itself (e.g., timer event) or in other ways (e.g., time triggered event). Typically, events are handled asynchronously with the program flow. The SMART**UNIFIER** software can also trigger its own set of events into the event loop, e.g., to communicate the completion of a task. Each event defined in an *Information Model* has an event type.

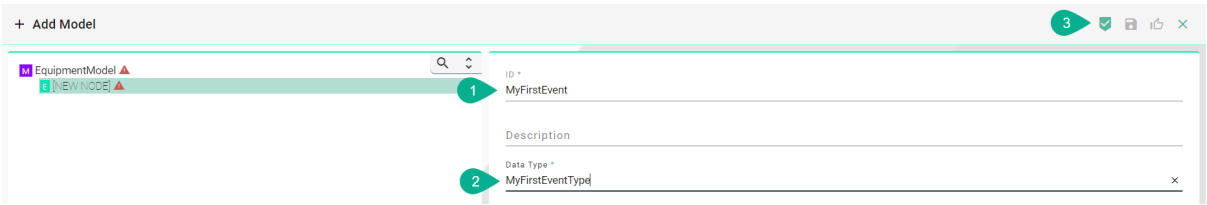
An event type consists of one or multiple simple or structured variables. Clients subscribe to such events to receive notifications of event occurrences.

Data Types

Event Node Types can be defined using *Custom Data Type*.

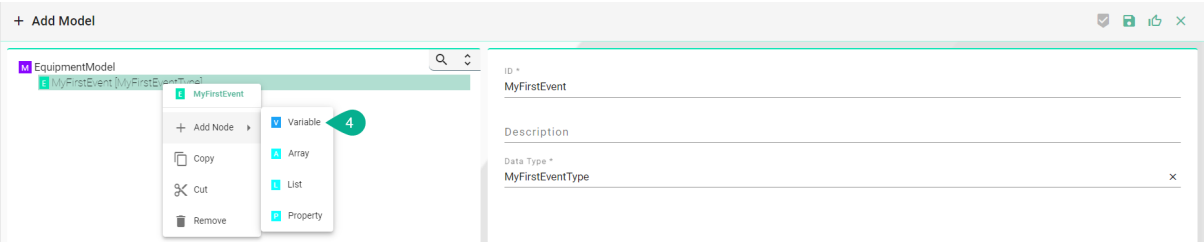
How to create an Event

- Enter an ID **(1)**
- Enter a Data Type for the Event. e.g., "MyFirstEventType" **(2)**
- Click the "Apply" button **(3)**

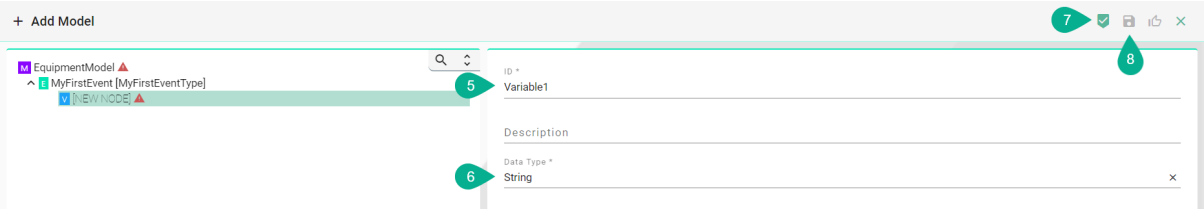


Within the Event *Variables*, *Arrays* or *Lists* can be added. Follow the steps below to add a Variable:

- Right click the Event node, select "Add Node" and choose a Definition Type **(4)**



- Enter an ID **(5)**
- Enter a Data Type **(6)**
- Click the apply button **(7)**
- Click the “Save” button at the top right corner **(8)** to save the Information Model



Commands

What are Commands

Commands are functions, whose scope is bound by an owning *Information Model*, like the methods of a class in object-oriented programming. Commands within an Information Model are typically invoked by an external IT system (e.g., an equipment) that triggers the command. In addition, commands of a target Information Model (e.g., an MES) can be triggered by the SMART**UNIFIER** through a *Mapping*. A command contains one or multiple simple or structured *Variables*. Also a command has a return parameter that likewise can be a simple or complex *data type*.

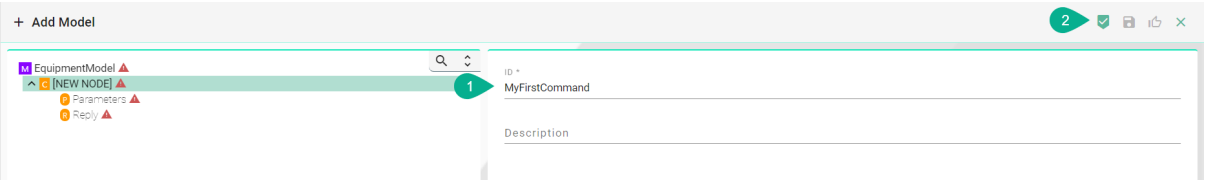
The lifetime of the command invocation instance begins when the client calls the command and ends when the result is returned. While commands may affect the state of the owning model, they have no explicit state of their own. In this sense, they are stateless. Each command defined in an Information Model has a command type.

Data Types

Command Node Types can be defined using *Custom Data Type*.

How to create a Command

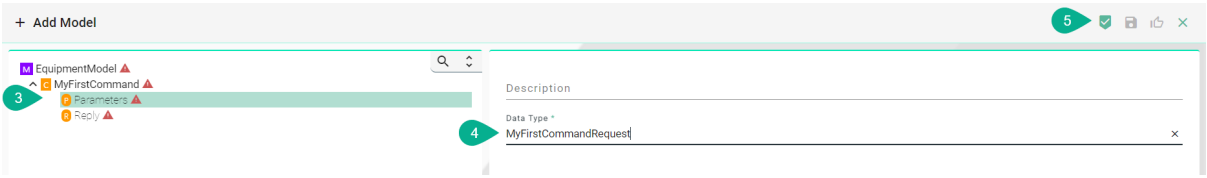
- Enter an ID **(1)**
- Click the "Apply" button **(2)**



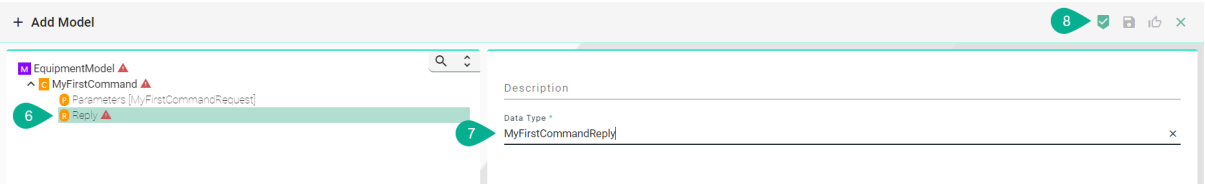
The main two parts of a Command are the Request, referred to as Parameters within the SMART**UNIFIER**, and the Reply. *Variables*, *Arrays* and *Lists* can be added to both of these command parts.

Follow the steps below to add a Variable to Parameters:

- Select the Parameters node from the tree **(3)**
- Enter a Data Type **(4)**
- Click the "Apply" button **(5)**

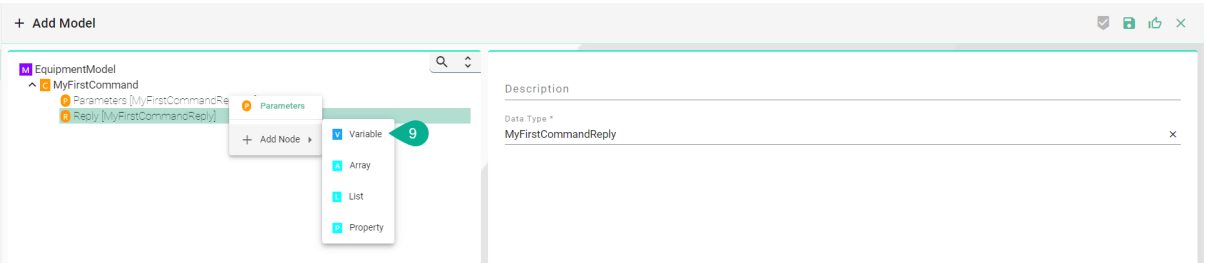


- Select the Reply node from the tree **(6)**
- Enter a Data Type **(7)**
- Click the "Apply" button **(8)**

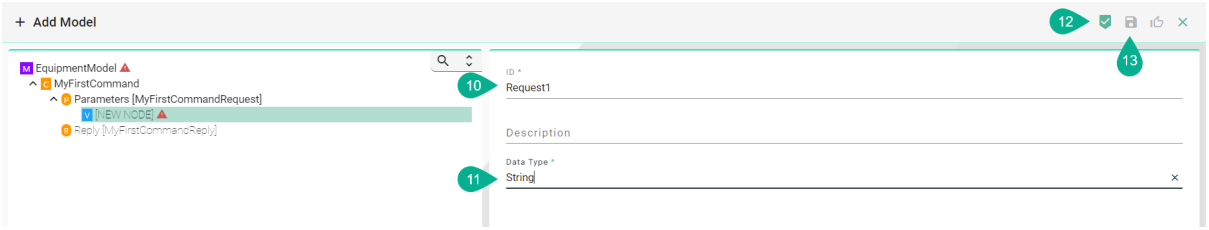


Follow the steps below to add nodes under the Parameter and Reply node:

- Right click the Parameter node, select "Add Node" and choose a Definition Type **(9)**



- Enter an ID **(10)**
- Enter a Data Type **(11)**
- Click the "Apply" button **(12)**
- Click the "Save" button **(13)** to save the Information Model



Arrays

What are Arrays

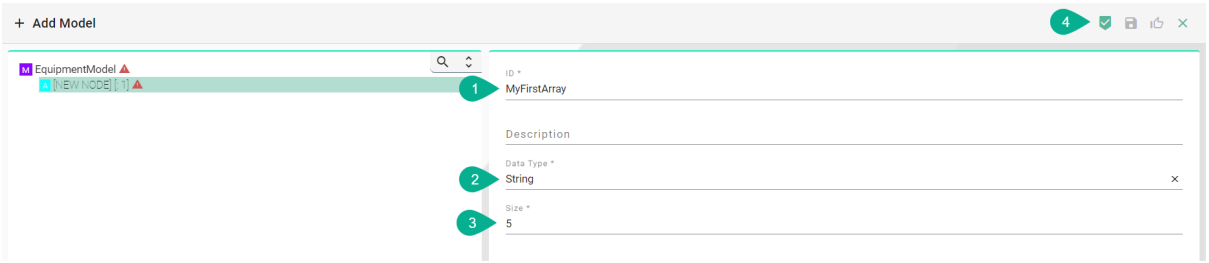
Arrays are designed to store a fixed number of elements, all of which share the same data type. The array's size needs to be specified in the Information Model's configuration. When defined as a *Custom Data Type*, arrays enable the creation of structured elements including *Variables*, *Arrays*, *Lists*, and *Properties*.

Data Types

- For Arrays containing values of a specific type, use the *Simple Data Type*.
- To create structures with other Note Types, use the *Custom Data Type*.

How to create an Array

- Enter an ID **(1)**
- Select a Data Type for the Array by clicking the Data Type Drop-Down **(2)**
- Enter the size of the Array **(3)**
- Click the "Apply" button **(4)**



Lists

What are Lists

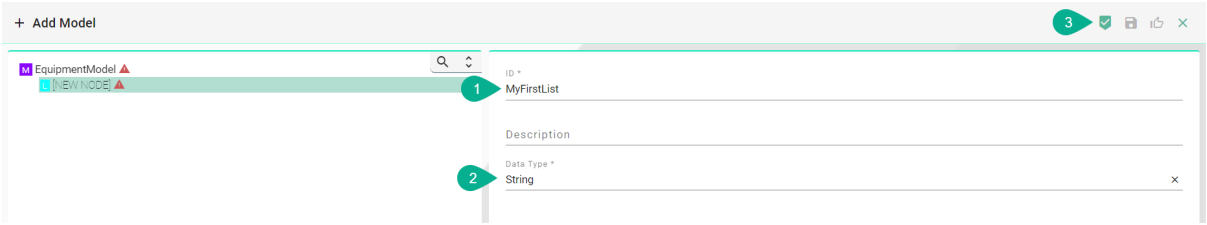
Lists allow to hold a collection of elements (*Variables*, *Arrays*, *Lists*, *Properties*).

Data Types

- For Lists containing values of a specific type, use the *Simple Data Type*.
- To create structures with other Note Types, use the *Custom Data Type*.

How to create a List

- Enter an ID **(1)**
- Enter a Data Type for the List. E.g., "String" **(2)**
- Click the "Apply" button **(3)**



Data Types

There are two different types of Data Types that can be used in the Information Model:

Predefined Types

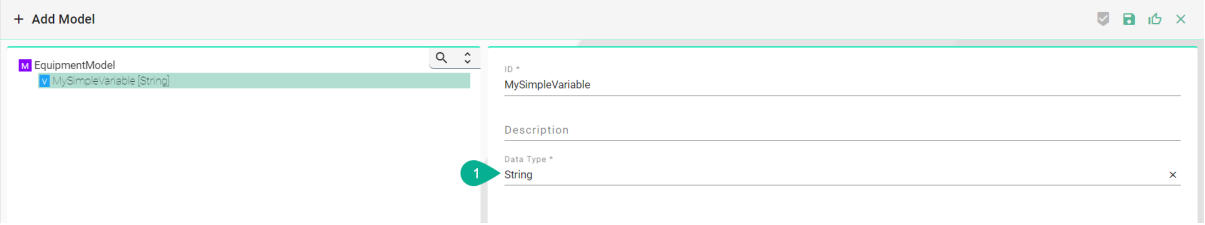
These are the standard (primitive) data types that are available in the SMART**UNIFIER**. They include,

Type	Definition
Boolean	true or false
Byte	8 bit signed value (-27 to 27-1)
Int	32 bit signed value (-231 to 231-1)
String	Sequence of characters
Char	16 bit unsigned Unicode character (0 to 216-1)
Double	64 bit IEEE 754 double-precision float
Float	32 bit IEEE 754 single-precision float
Long	64 bit signed value (-263 to 263-1)
Short	16-bit signed integer
LocalDate-Time	Immutable date-time object that represents a date-time, often viewed as year-month-day-hour-minute-second.
Offset-DateTime	Immutable representation of a date-time with timezone information.

Data types like these can be applied to the following Node Types: Variables, Properties, Arrays, Lists.

Example - Variable with a Predefined Data Type:

- Add a new Variable
- Enter an ID (Name)
- select a predefined data type e.g., "String" **(1)**



Custom Types

Custom data types are user-defined data types that can be created and used in the Information Model. They can be applied to the following Node Types: Variables, Properties, Arrays, Lists, Events, Commands. They come in handy, for example, when a complex data structure is required.

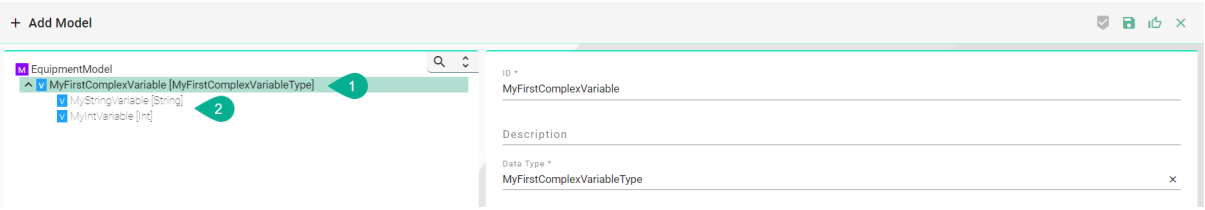
The name of the Data Type is to be defined by the user. It is recommended to name custom data types in a way that reflects the data structure they represent.

Example - Variable with a Custom Data Type:

- Add a new Variable
- Enter an ID (Name) e.g., "MyFirstComplexVariable"
- Enter a custom name for the Data Type e.g., "MyFirstComplexVariableType" **(1)**

Now another Variable can be added under the "MyFirstComplexVariable" - Variable:

- Perform a right click on the Custom Variable - "MyFirstComplexVariable"
- Select "Add Node" and choose Variable **(2)**



Hint

Model *Node Types* with custom data types can be easily duplicated throughout the Information Model by selecting the same custom data type for a new model node type.

Structures Required by Channels

The structure of an Information Model is influenced by the Communication Channel used in the integration. Communication Channels can be classified into two types: data-driven and event-driven communication.

- **Event-driven** involves integration triggered by an event within a system, such as the receipt of goods. This event activates a Rule within the Mapping of SMART**UNIFIER**.
- **Data-driven**, on the other hand, is initiated by a change in the data within a system.

- Additionally, there is **command-driven** integration, where an event in one system immediately requires a response from another system. This differs from event-driven integration, which does not anticipate a direct reply.

Below, a table presents some examples of use cases along with the Information Model structures they require:

Use Cases	Description	Communication Channel	Information Model Structure
Data-driven	Retrieving data from OPC-UA Server	OPC-UA Client	Structure of Variables (Predefined and Custom Data Types) representing the data structure of the OPC-UA Server
Event-driven	Posting data on MQTT Broker	MQTT client	Using Events with a structure of Variables (both Predefined and Custom Data Types) to represent the data structure of, for example, a JSON message.
Command-driven	Executing a Select request on a database with parameters.	SQL Database	Using Commands with structures of variables under *Parameters* and *Reply*

Hint

The required structure for the Information Model corresponding to each *Communication Channels* is described in the specifc Channel Type documentation.

Importing Data Structures

Data structures can be imported by using extensions, which is especially convenient when dealing with complex structures that include many variables. The following import options are available:

- *OpcUa Model Import* : Data structures can be imported from an OPC UA server.
- *JSON Model Import*: JSON structures can be imported directly.

Shortcuts

Shortcut	Action
CTRL + C	Copy the selected object
CTRL + V	Paste the object from the clipboard
ArrowUp	Navigate up
ArrowDown	Navigate down
ArrowLeft	Navigate up or collapse node
ArrowRight	Expand node
ALT + N	Open a new contextmenu
CTRL + X	Cut the selected object
CTRL + D	Duplicate the selected object
CTRL + DELETE	Delete the selected object

Communication Channels

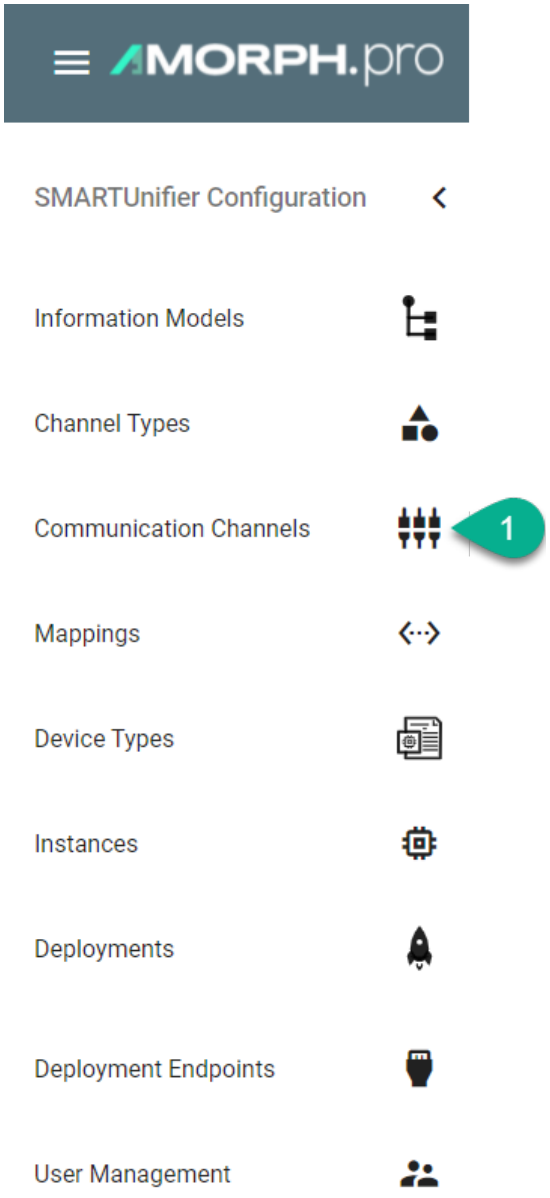
What are Channels

Communication Channel or simply Channel refers to a transmission medium. A Channel is used to convey information from one or several senders (or transmitters). Communicating data from one location to another requires a pathway or medium. These pathways are called Communication Channels, and the information is transmitted with the help of communication protocols. Each *Information Model* can have one Channel or many, and each model can choose which Channels it subscribes to. The information is transmitted through the Communication Channels in both directions: from the external system to the SMART**UNIFIER** application and vice versa.

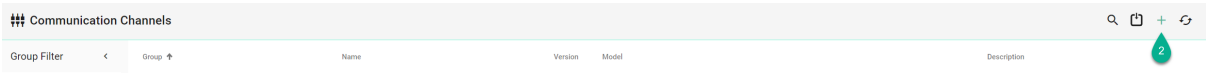
How to create a new Channel

Follow the steps below to create a new Channel:

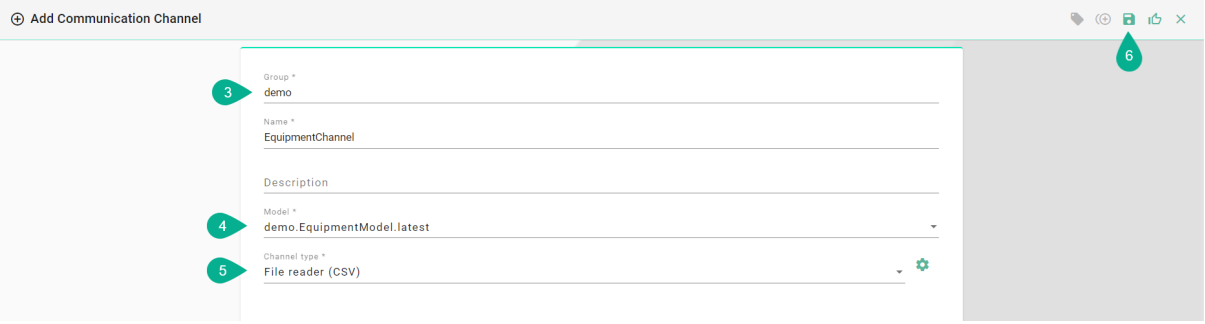
- Go to the Communication Channels perspective by clicking the "Communication Channels" button **(1)**



- To create a new Channel, select the "Add Channel" button at the top right corner **(2)**



- The creation of a Communication Channel is split up into two parts. First enter basic information about the new Communication Channel
  - Fill in the information for the Channel identifier such as: Group, Name and Version. Description is optional **(3)**
  - Besides that, associate the Channel with an Information Model **(4)**
  - Select the type this Channel represents from the Drop-Down **(5)** - A list of available Channel Types and a description of how to configure each of them can be found below
- Click the "Save" button **(6)** to save the Channel



Channel Types and Configuration

There are several Channel types available with SMART**UNIFIER**. A listing of Communication Channel types can be found at https://amorph.pro/getsmartertopic/smart-connectivity/. If a specific Communicating Channel type is not available in this product version, please contact Amorph Systems. In many cases the provision of a specific Communication Channel type can be provided as extension to the standard product.

The configuration of the Communication Channels can be done on Channel, *Device Type* and *Instance* level.

> **Note**
>
> It's important to note that the configuration of a Channel can be overwritten as needed. For example, the configuration made in the Communication Channel view can be changed in the Instance view.

The following paragraphs lay out the configuration process of selected Channel Types. If the Channel Type you want to use is not described, please contact Amorph Systems for configuration guidance.

File-based

File Reader

Characteristics

- File Reader monitors a specified folder - the so-called input folder
- If a file is inserted the following actions take place:
  - The *Trigger* of the specified *Rule* in the Mapping is activated
  - Thus, the Rule is executed
- After successful execution of the rule the file is moved into a so-called output folder
- In case of an exception the file is moved into an error folder
- The File Reader can be used for different file formats like *CSV*, *JSON* and *XML*

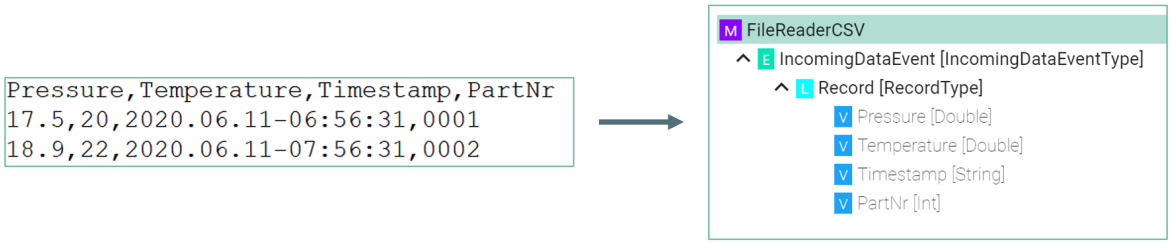
Information Model Requirements

The first Node after the root node M must be of type Event E. Depending on the file format, the Information Model needs to be built up accordingly:

CSV

- Reading multiple lines from a CSV file requires a *List* L.
- Data in the CSV columns is represented by the Node Type *Variable* V. Note that the order of fields in the CSV file must match the order of Variables in the Information Model.

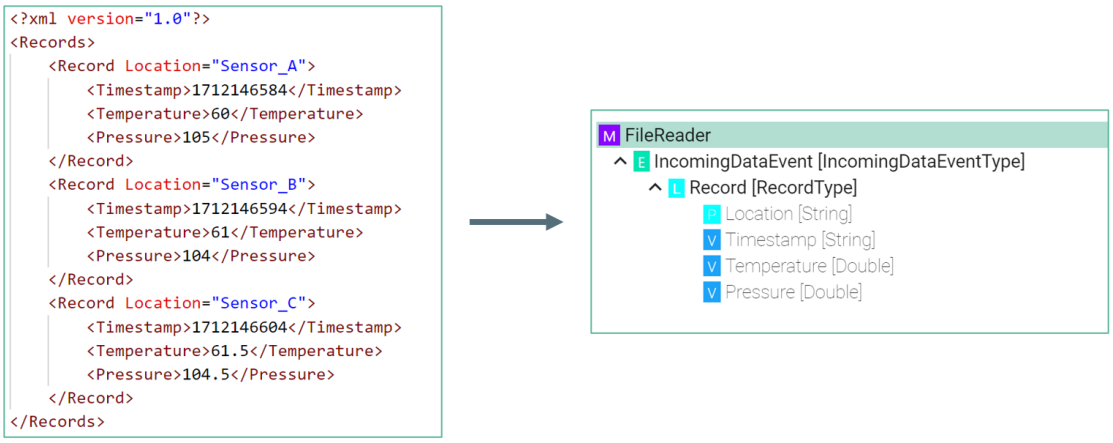
Below is an example of a CSV file containing multiple records and the corresponding Information Model:



JSON

- Reading a JSON file containing a list structure requires a *List* L.
- Keys are represented by the Node Type *Variable* V.

Below is an example of a JSON file containinga list and the corresponding Information Model:

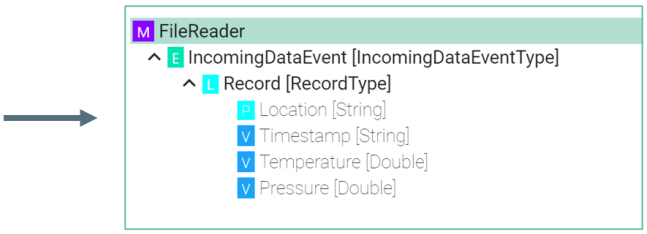


XML

- Elements of the XML file are represented by the Node Type Variable V.
- Attributes of the XML file are represented by the Node Type *Property* P. In order to assign attributes to elements in the Information Model, the element Node Type V must be a *Custom Data Type*.

Below is an example of a XML file containing multiple records and the corresponding Information Model:

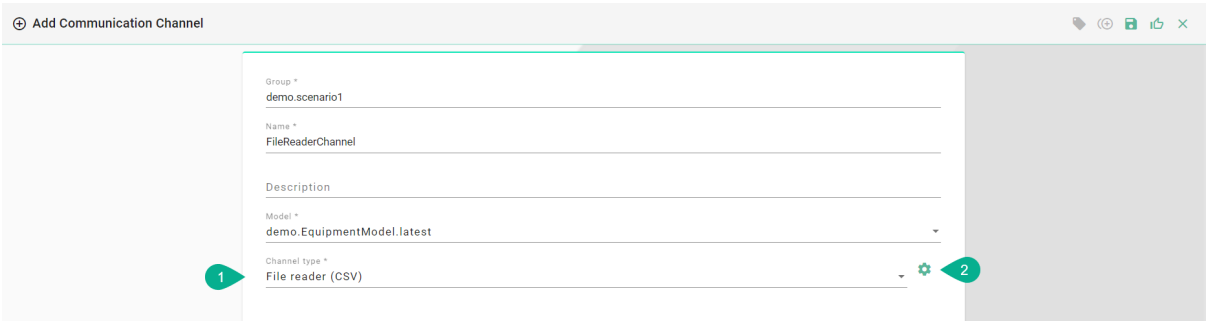
```
<?xml version="1.0"?>
<Records>
    <Record Location="Sensor_A">
        <Timestamp>1712146584</Timestamp>
        <Temperature>60</Temperature>
        <Pressure>105</Pressure>
    </Record>
    <Record Location="Sensor_B">
        <Timestamp>1712146594</Timestamp>
        <Temperature>61</Temperature>
        <Pressure>104</Pressure>
    </Record>
    <Record Location="Sensor_C">
        <Timestamp>1712146604</Timestamp>
        <Temperature>61.5</Temperature>
        <Pressure>104.5</Pressure>
    </Record>
</Records>
```



Configuration

In this example, the setup of the File Reader for CSV is illustrated.

1. Select the File Reader with the file format to be used from the Drop-Down.
2. Click the **Configure** button.

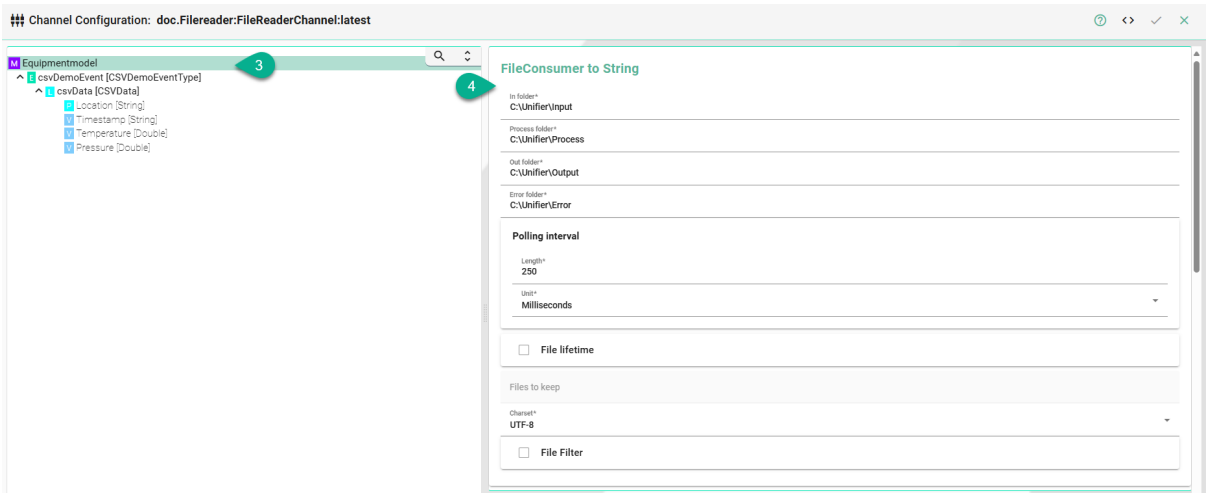


3. Ensure the root model node is selected to configure the File Consumer to String as well as the CSV String to Model.
4. File Consumer to String - Configuration
    - Enter a path for the input folder - **In Folder**
    - Enter a path for the process folder - **Process Folder**
    - Enter a path for the output folder - **Out Folder**
    - Enter a path for the error folder - **Error Folder**

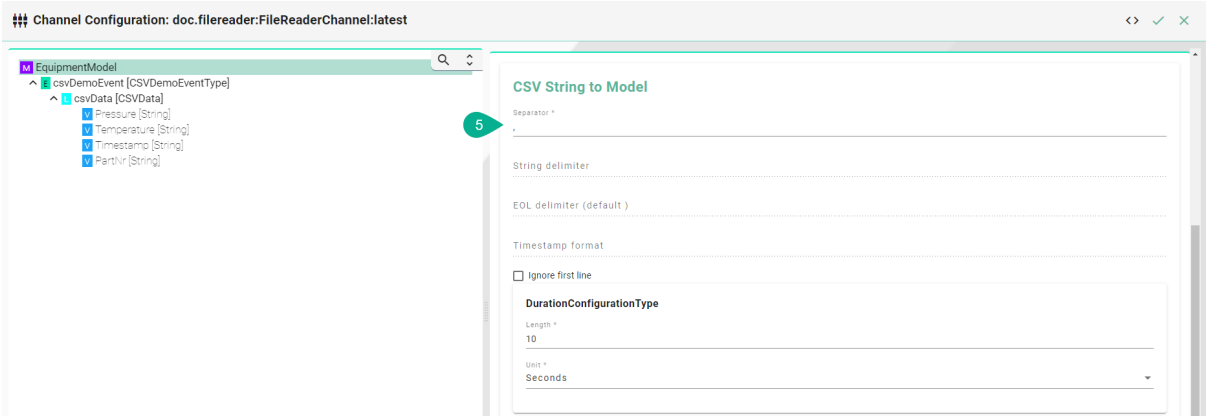
- Specify the **Polling interval** and select the **Unit**
- (Optional) Select the checkbox **File lifetime** to delete files after the set time in the output and error directories.
- (Optional) Enter the number of **files to keep** in the output and error directories.
- Select the **CharSet** according to the file in use
- (Optional) Select the **File Filter** checkbox to limit the file size; files larger than the threshold are moved to the unprocessed directory.

Note

If **Files to Keep** is used with **File Lifetime**, this value takes precedence in the cleanup process, meaning older files are not deleted until the threshold is reached.



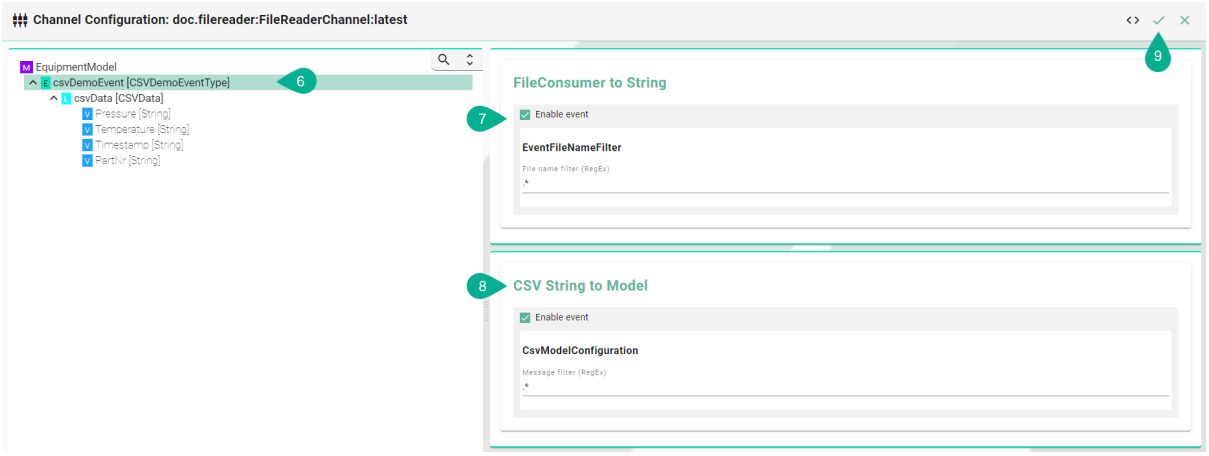
5. *CSV String to Model* - Configuration
   - Enter the **Separator** which is used in the CSV-file
   - If needed, set **String delimiter**, **EOL delimiter** and the **Timestamp format**
   - If the CSV file contains a header enable **Ignore first line**
   - Specify the **Polling interval** and select the **Unit**



Note

To configure *XML to Model* or *JSON to Model* navigate to the respective sections.

6. Specify the Event used by selecting the **event node** in the tree on the left side
7. File Consumer to String - Configuration
   - Enable the **Event** checkbox for the **File Name Filter**
   - Enter a **Regular expression** in order to determine which file is to be processed in the input folder
8. *CSV String to Model* - Configuration
   - Enable the **Event** checkbox for the **Csv Model Configuration**
   - Start of processing
     - If the entire content of the file is processed on this event enter a wildcard in the **RegEx** field
     - If the processing starts at a specific line enter a regular expression in the **RegEx** field to identify the line
9. Click the **Apply** button, then the **Close** button and save the Channel by clicking the **Save** button



Description of configuration properties:

Property	Description	Example
Separator	Separator type, used in the csv file	, , ;
Delimiter	Values that have an additional delimiter like "Date", "Time"	"
Eol Delimiter	Defining Carriage return and/or Line Feed	\r, \n
Timestamp format	Format of the timestamp	YYYY-MM-DD HH:mm:ss
ignoreFirstLine	Delay between checks of the file for new content in milliseconds	true, false
TailFromEnd	Set to true to tail from the end of the file, false to tail from the beginning of the file	true, false
InFolder	Path leading to the Input Folder	C:\FileConsumer\In
OutFolder	Path of a node in the Information Model	C:\FileConsumer\Out
ErrorFolder	Regular Expression for the message filter used in the implementation	C:\FileConsumer\Error
CharSet	Encoding of the file in use	UTF-8, UTF-8 BOM, etc
ProcessFolder	Regular Expression for the message filter used in the implementation	C:\FileConsumer\Process

File Tailer

Characteristics

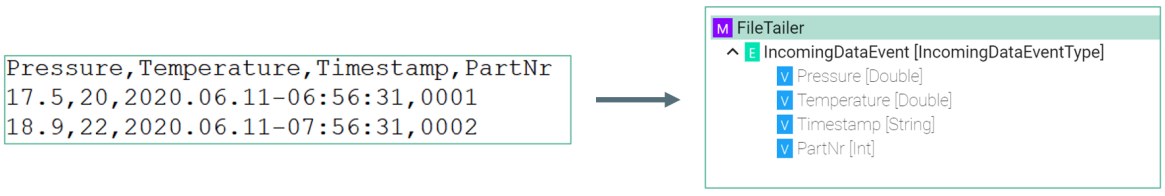
- The File Tailer monitors a given file in a specified location.
- Data is processed line by line.
- Note that the File Tailer does not support the node type **List** in the *Information Model*.
- The File Tailer can be used for *CSV* Files.

Information Model Requirements

The first Node after the root node M must be of type Event E.

CSV

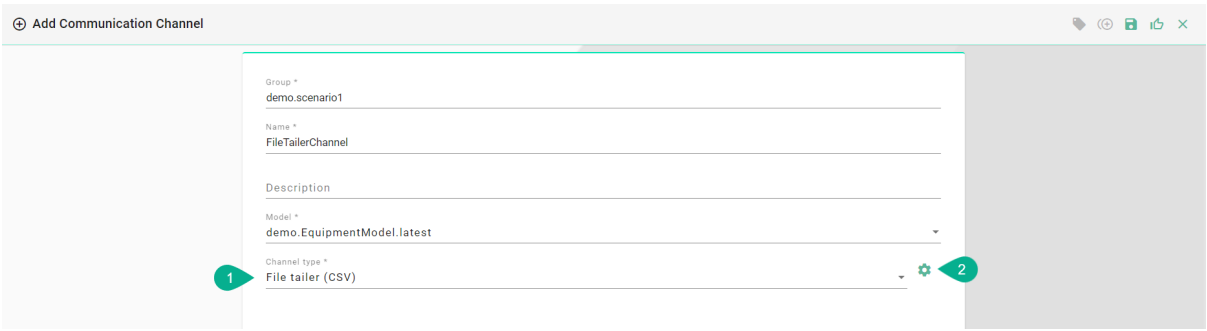
- Data in the CSV columns is represented by the Node Type *Variable* V. Note that the order of fields in the CSV file must match the order of Variables in the Information Model.



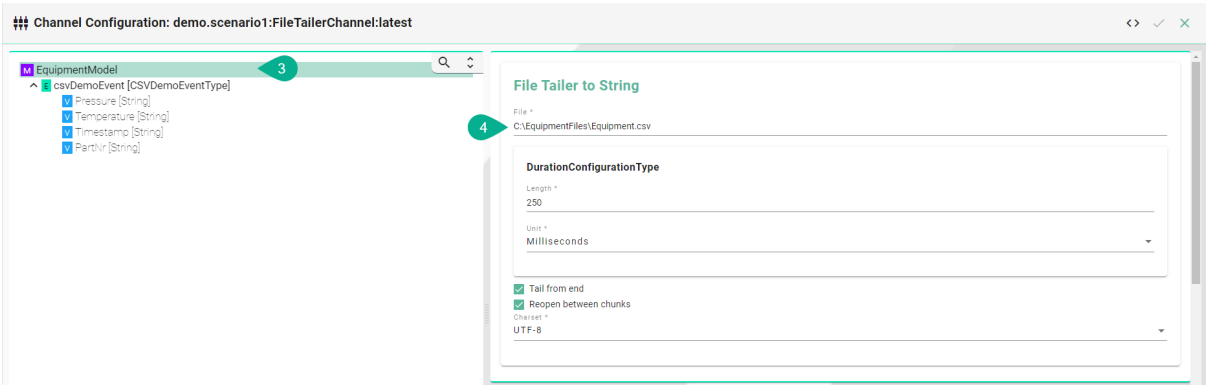
Configuration

In this example, the setup of the File Tailer for CSV is illustrated.

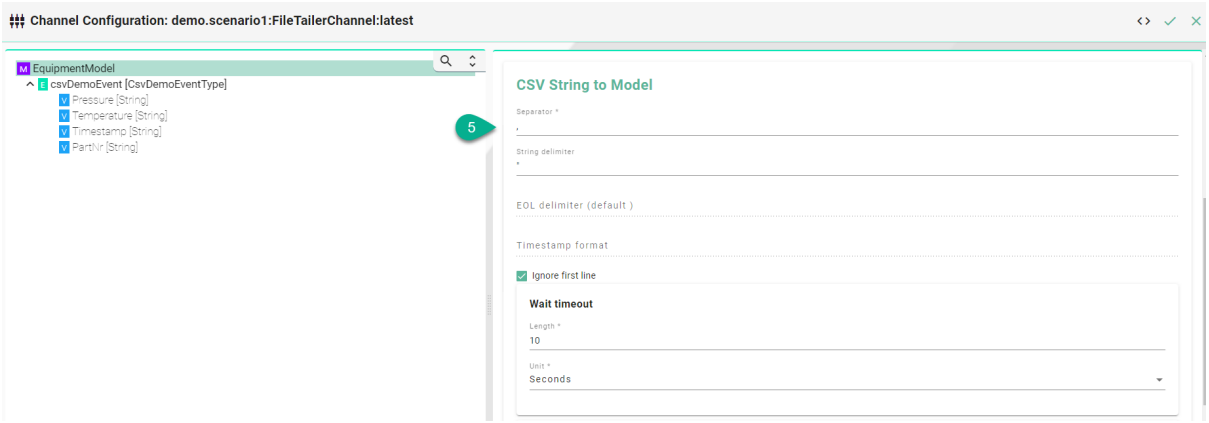
1. Select **File tailer (CSV)** from the Drop-Down.
2. Click the **Configure** button.



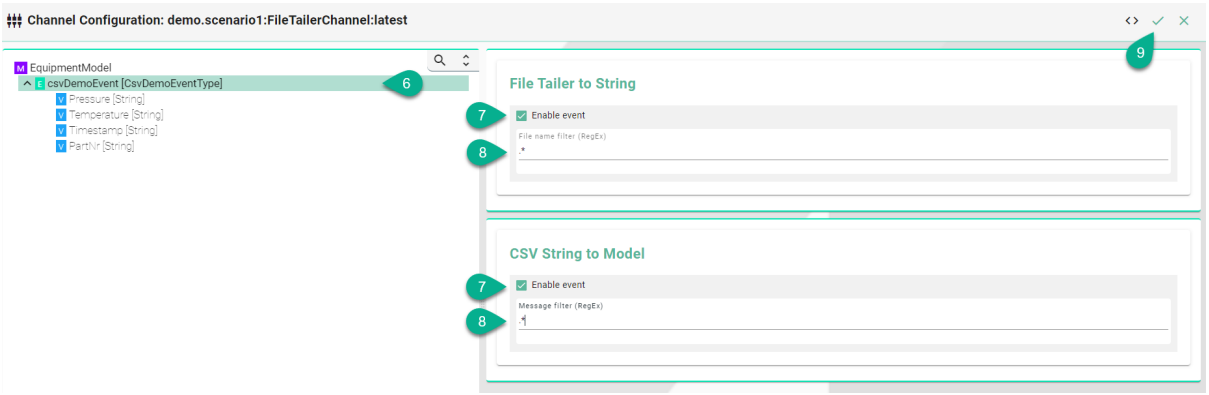
3. **Make sure** the root model node is selected to be able to configure the File Tailer to String and CSV String to Model.
4. File Tailer to String - Configuration:
   - Enter the **File path** for the CSV-file on your machine
   - Specify the **Polling interval** and select the **Unit**
   - Enable **Tail from end** if you want to pick up always the last line of the file
   - Enable **Reopen between chunks** if the file should be closed and reopened between chunks
   - Select the **Charset** according to the file in use



5. *CSV String to Model* - Configuration:
   - Enter the **Separator** which is used in the CSV-file as well as the **String delimiter**
   - Input the **Eol delimiter** and the **Timestamp format** if one is used.
   - If the CSV file contains a header enable **Ignore first line**
   - Input the **Polling interval** and select the **Unit**



6. Select the **event node** in the tree on the left side.
7. Check the **Routes checkbox**.
8. Enter a **Regular expression** for the message filter.
9. Click the **Apply** button, then the **Close** button and save the Channel by clicking the **Save** button on the upper right corner.



Description of configuration properties:

Property	Description	Example
Separator	Separator type, used in the csv file	`,` , `;`
Delimiter	Values that have an additional delimiter like "Date", "Time"	`"`
Eol Delimiter	Defining Carriage return and/or Line Feed	`\r`, `\n`
Timestamp format	Format of the timestamp	`YYYY-MM-DD HH:mm:ss`
File	Path to the csv file	`C:\test.csv`
Delay Millis	Delay between checks of the file for new content in milliseconds	`250`
TailFromEnd	Set to true to tail from the end of the file, false to tail from the beginning of the file	`true`, `false`
ReopenBetweenChunks	If true, close and reopen the file between reading chunks	`true`, `false`
routes	Path of a node in the Information Model	`true`, `false`
messageFilterRegEx	Regular Expression for the message filter used in the implementation	`.*`

File Writer

Characteristics

- The File Writer writes data to a file in CSV, JSON, or XML format.
- The data can be written to the file in either append mode or overwrite mode.
- The File Writer can be used for different file formats like *CSV*, *JSON* and *XML*

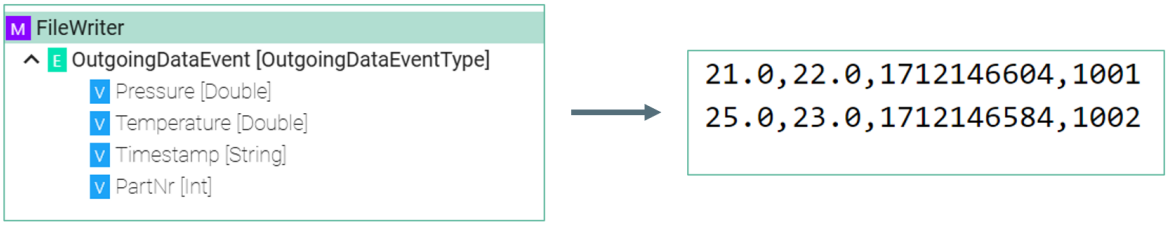
Information Model Requirements

The first Node after the root node M must be of type Event E.

CSV

- CSV columns are represented by the Node Type *Variable* V. Note that the order of the Variables in the Information Model determines the order of the columns in the produced CSV file.

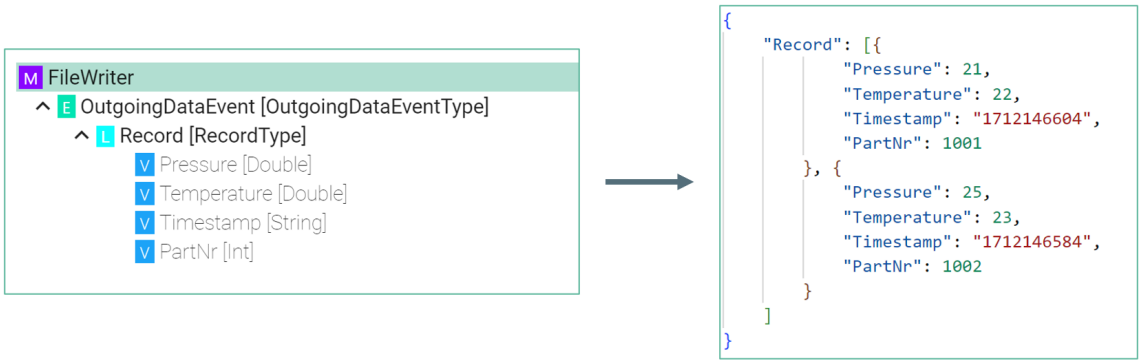
Here is an example of the Information Model and how the data is written to the CSV file:



JSON

- The key-value pairs are represented by the Node Type *Variable* V.
- Objects require *Variables* V of *Custom Data Type*.
- Writing list structures to a JSON file requires a *List* L.

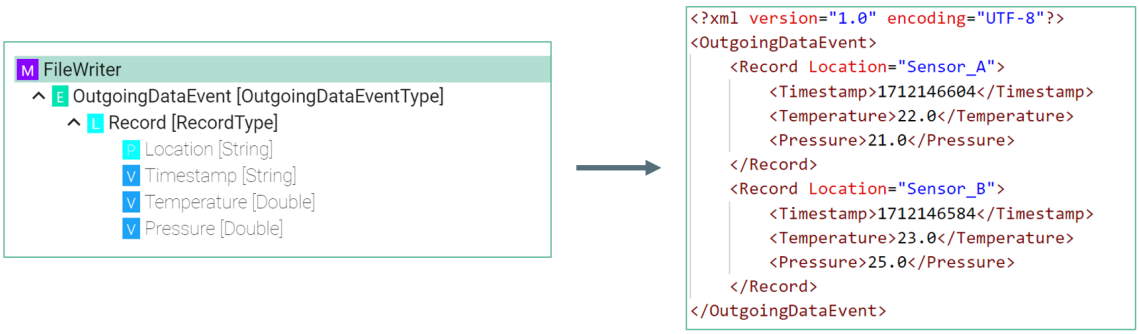
Here is an example of the Information Model and how the data is written to the JSON file:



XML

- Attributes of the XML file are represented by the Node Type *Property* P.
- In order to assign attributes to elements in the Information Model, the element Node Type V must be a *Custom Data Type*.
- Writing list structures to a XML file requires a *List* L in the Information Model.

Here is an example of the Information Model and how the data is written to the XML file:

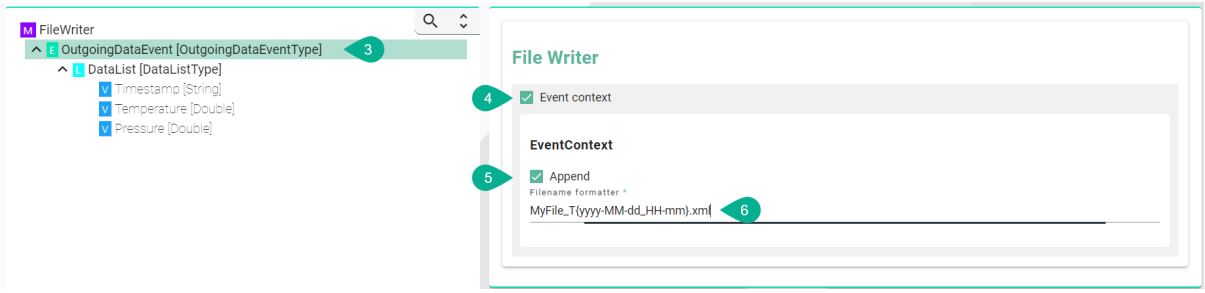


Configuration

1. Ensure the root model node is selected in the Information Model.
2. Enter a **path** for the location where the file should be written.

> **Note**
>
> If a path is entered without any slashes, it is interpreted as the name of a directory to be created within the deployed Instance.

3. Select the **Event** node in the Information Model.
4. Enable the checkbox **Event context**
5. (Optional) Enable **Append** to append data to the file. If this option is disabled, the existing file will be overwritten.
6. Enter a **Formatter** for the file name. Placeholders can be used to dynamically name the file. A placeholder can be a value from the variable itself, such as `${Timestamp}`, or it can be a direct timestamp format, such as `T{yyyy-MM-dd_HH-mm}`.



Note

If the file name changes, a new file will be created. For example, if the minute changes and a timestamp is part of the formatter, a new file will be generated.

Databases

SQL Database

Characteristics

- The SQL Channel can be configured for the following two scenarios:
  - Inserting data
  - Updating data
  - Retrieving data
- When inserting values into the database please **note** that "infinity" values are converted automatically into "null" values.

Supported features

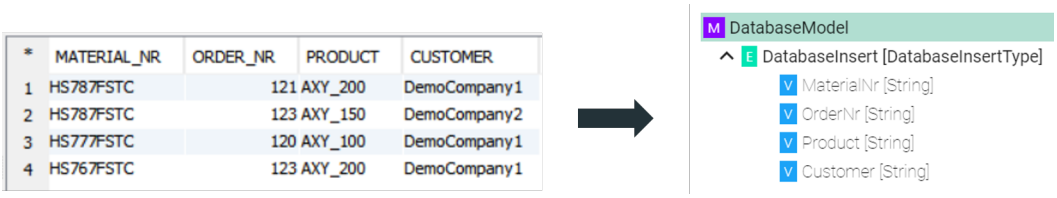
- Supported databases
  - DB2
  - HSQLDB
  - ORACLE

  - PostgreSQL
  - SQLServer
  - MariaDB / MySQL
- Custom queries

Information Model Requirements

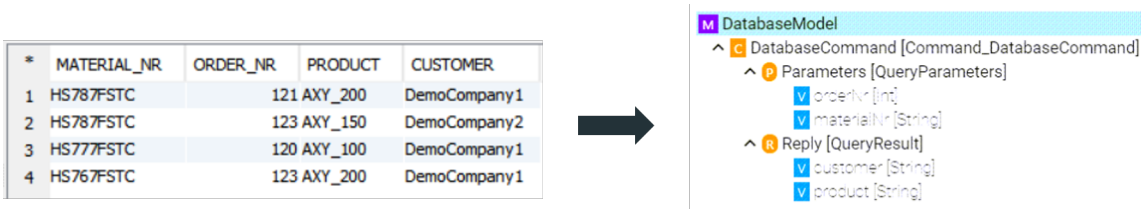
Insert/Update

- The node after the root model node must be of type *Event* E which represent a database table.
- In case of relational databases: Tables which are dependent on each other require a *List* L.
- Columns of databases are represented by *Variables* V.



Select

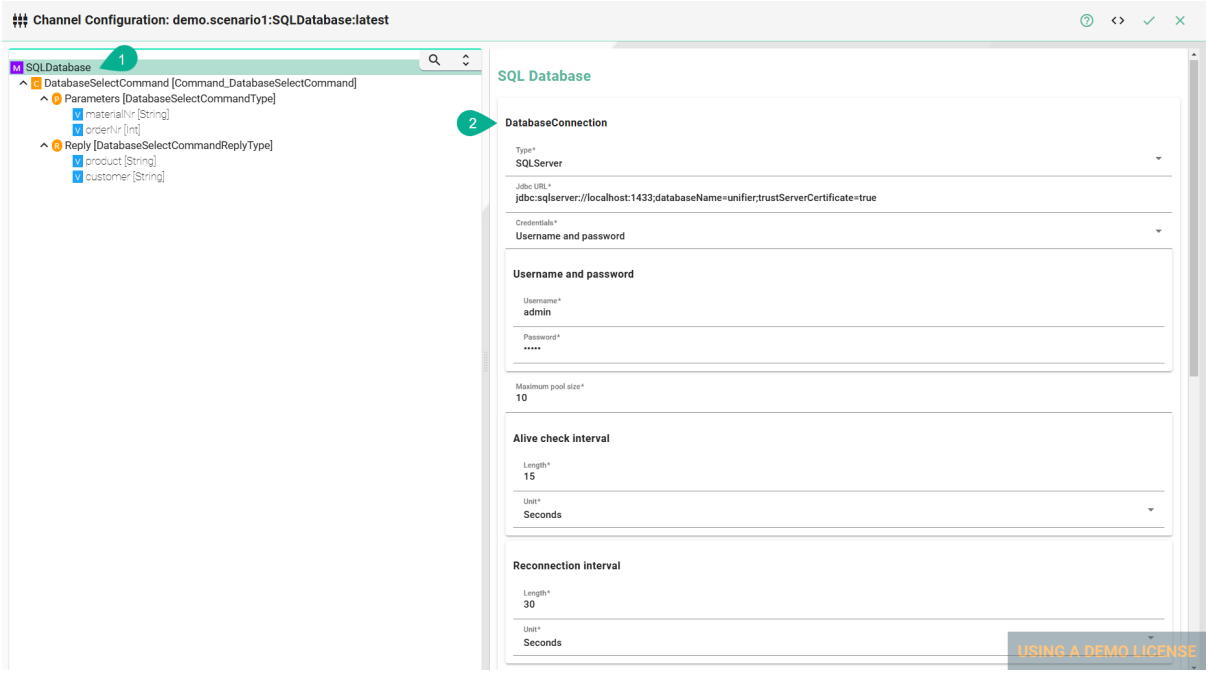
- The *Command* C defines that after a request is made, a reply with a result is expected.
- Parameters P within a Command represent a collection of query parameter - query parameters are defined as Variables V.
- Reply R within a Command represents the result of the Command - results are defined as Variables V.



Configuration

1. Select the **root model node** in the tree on the left.
2. Configure the database connection
   - Select the **Database type**
   - Enter the **database connection URL** for the specific database type
     - DB2: `jdbc:db2:server:port/database`
     - HSQLDB: `jdbc:hsqldb:file:databaseFileName;properties`
     - ORACLE: `jdbc:oracle:thin:prodHost:port:sid`

  - PostgreSQL: `jdbc:postgresql://host:port/database`
  - SQLServer: `jdbc:sqlserver://[serverName[\instanceName][:portNumber]][;property=value[;property=value]]`
  - MariaDB: `jdbc:(mysql|mariadb):[replication:|loadbalance:|sequential:|aurora:]//<host>[:<portnumber>]/[database][?<key1>=<value1>[&<key2>=<value2>]]`
- Enter the database **Username** and **Password** or select it from the Credentials Manager
- Set the **Maximum pool size**
- Specify the **Alive check interval**
- Specify a **Reconnection interval**



Description of configuration properties:

Table 1: Database Properties

Property	Description	Example
Type	Type of the database	`MariaDB`, `SQLServer`, `ORACLE`, `HSQLDB`, `DB2`, `PostgreSQL`
ReconnectInterval	Time to reconnect if connection to the database fails	`10` (in milliseconds)
JdbcUrl	Url to connect to database	• jdbc:sqlserver://localhost:1433;databaseName=unifier;trustServerCertificate=true • jdbc:mariadb://localhost:3306/unifier?connectTimeout=5000 • jdbc:db2://127.0.0.1:50000/TESTDB • jdbc:hsqldb:file:\protect\T1\textdollardbFileName;shutdown=true • jdbc:postgresql://127.0.0.1:5432/postgres • jdbc:oracle:thin:@localhost:1521/MYCDB - See the *Oracle info*
Username and **password**	Credentials of the database	
Alive check interval	Duration between checks to determine if a database connection is still alive	e.g.: `15` (seconds)
Maximum pool size	Specifies the number of connections that can be maintained in the pool (ensuring that the database can handle a specified number of simultaneous database interactions)	e.g.: `10`

Note

The configuration of specific *information model nodes* differs whether you want to perform an **insert** or an **select** statement on the database. Inserting data into the database requires an **event node** whereas selecting data requires a **command node** in the *Information Model*.

ORACLE

The JDBC Driver by default is not included in the SMART**UNIFIER** package. To use the database type Oracle please follow these steps:

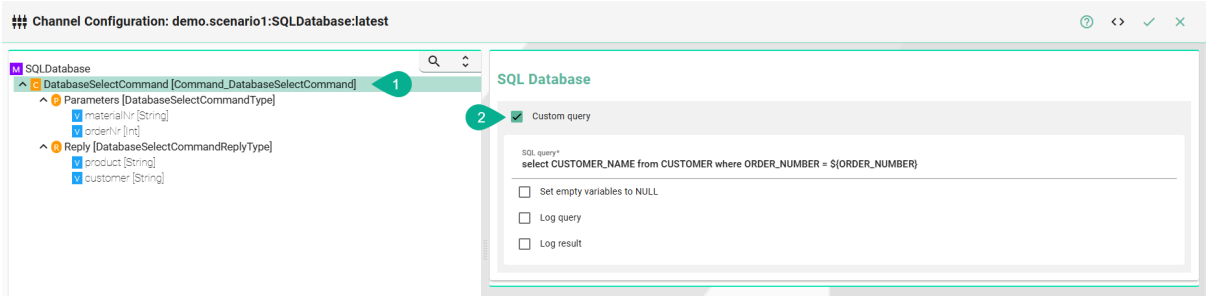
1. Review and accept the Oracle License Agreement
2. Navigate to **..SmartUnifierManager/repository/amorphsyslib/com.oracle.database.jdbc/ojdbc11/jars**
3. Download the Oracle JDBC Driver and add it to the specified directory

Select Statement

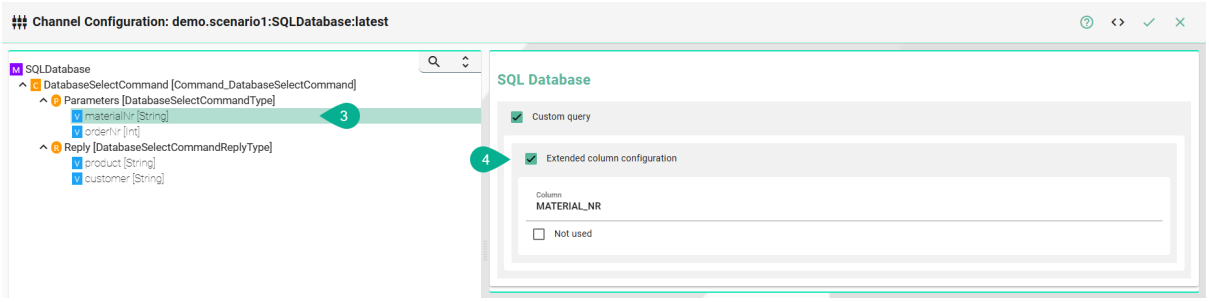
1. Select the **command node** in the tree on the left.
2. Check the **Custom Query** checkbox
   - Enter the **SQL Query**
   - (Optional) Check the **Set empty variables to NULL** if emtpy variables should be set to *NULL* in the SQL statement
   - (Optional) Check the **Log query** checkbox to explicitly allow logging of the defined SQL statement to the database.
   - (Optional) Check the **Log result** checkbox to explicitly allow logging the entries (result) the query returned.

Note

Both the logging options (Log query and Log result) are recommended settings during the configuration and testing of the communication instance. In production mode, this should be disabled.

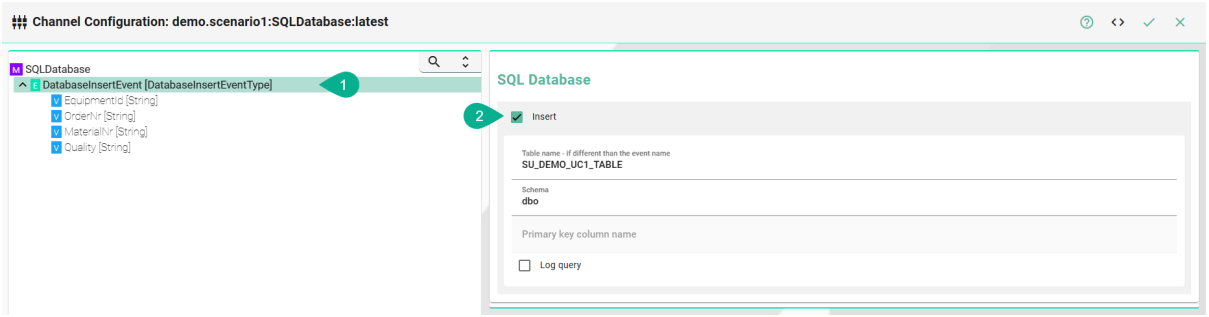


3. Each variable under *Parameters* and *Reply* needs to be assigned to a corresponding database column. To configure this, select the **variable node** under *Parameters* and then choose the appropriate option in the tree structure.
4. Check the **Assign database column** checkbox
   - Enter the **Column name** as it is defined in the used database
   - (Optional) Check **Not used** to exclude the variable from the database operation



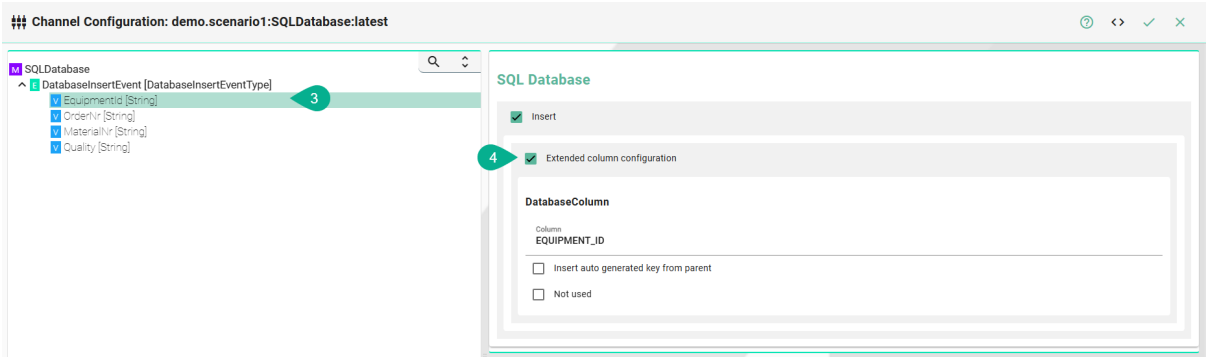
Insert Statement

1. Select the **event node** in the tree on the left.
2. Check the **Insert** checkbox
    - Enter the **Table name**
    - (Optional) If required enter a **Schema name**
    - (Optional) Enter the **Primary key column name**
    - (Optional) Check **Log query** to explicitly allow logging of the defined SQL statement to the database. This is recommended during the configuration and testing of the communication instance. In production mode, this should be disabled.



3. Select the **variable node** in the tree on the left
4. Check the **Assign database column** checkbox
    - Enter the **Column name**
    - (Optional) Check the **Insert auto generated key from parent** checkbox if the column relates to its parent
    - (Optional) Check **Not used** to exclude the variable from the database insert operation

> **Note**
>
> Configuration of the column name is only necessary if the column name in the database is different compared to the variable defined in the Information Model.



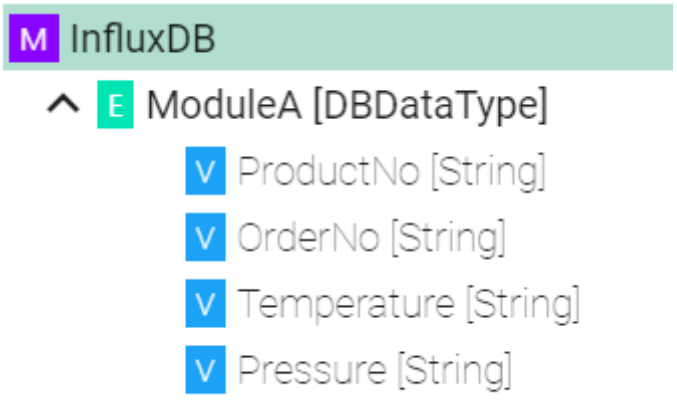
InfluxDB v1

InfluxDB v1 is the initial version of the high-performance time-series database designed for time-stamped data storage and real-time analytics, for more information visit the influxdata website.

Information Model Requirements

Writing

- Measurements are represented by *Event* E and Complex *Variables* V
- Fields are represented by *Variables* V by default
- Tags and Time are as well represented by *Variables* V but they have to be specifically configured (see below *Tags configuration* and *Time configuration*)

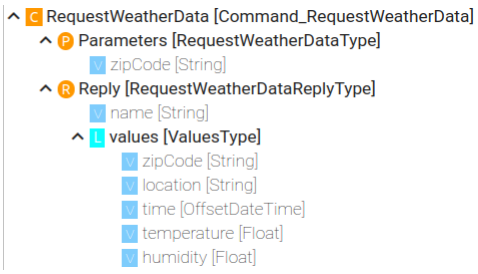


- Arrays A can be used to set use an index

Reading

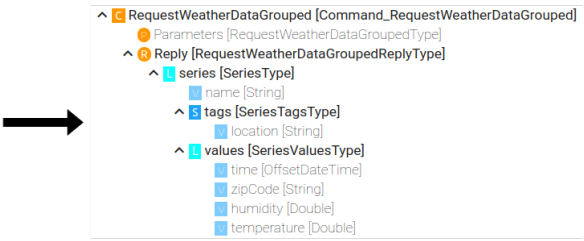
- Reading from InfluxDB is done using *Command* C, Complex *Variables* V and *Lists* L
- Result variables are optional, so the number of variables can be fewer than those returned by the query
- For the time the types string, long and OffsetDateTime are supported
- Variables in the requests can be used as parameters for building the query using the `${MyVariableName}` pattern
- For ungrouped result the following command reply structure needs to be used

```
> SELECT * FROM "Weather" WHERE  time >= now() - 20s ORDER BY time ASC
name: Weather
time                humidity           location  temperature        zipCode
----                --------           --------  -----------        -------
1752567115782487000 68.470947265625    LocationA 21.54784393310547  7154
1752567119782624000 58.52103042602539  LocationB 27.041414260864258 4343
1752567123782794000 67.75028991699219  LocationB 27.426149368286133 4343
1752567127782919000 53.928497314453125 LocationA 28.431154251098633 7154
1752567131783102000 69.3249740600586   LocationB 20.949594497680664 4343
```



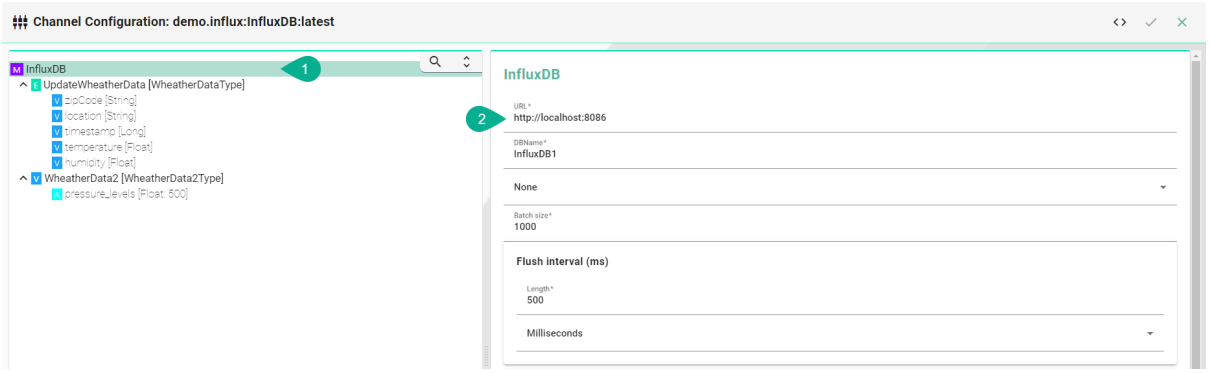
- For grouped result the following command reply structure needs to be used

```
> SELECT * FROM "Weather" WHERE  time >= now() - 20s GROUP BY "location"::tag ORDER BY time ASC
name: Weather
tags: location=LocationA
time                humidity           temperature        zipCode
----                --------           -----------        -------
1752566179742150000 61.93437957763672  22.298673629760742 4343
1752566183742804000 54.274532318115234 20.304012298583984 4343

name: Weather
tags: location=LocationC
time                humidity          temperature       zipCode
----                --------          -----------       -------
1752566171741872000 61.24681854248047 27.42499351501465 7154
1752566187743166000 54.62199401855469 22.06602668762207 1235

name: Weather
tags: location=LocationD
time                humidity           temperature       zipCode
----                --------           -----------       -------
1752566175742290000 50.437957763671875 26.12150764465332 1235
```



Configuration

1. Select the **root model node** in the tree on the left.
2. Configure the InfluxDB.
   - Enter the **URL** to the database
   - Enter the **Database name**
   - Enter the database **Username** and **Password** or select it from the Credentials Manager
   - Enter the **Batch size** - writes data in batches to minimize network overhead when writing data to InfluxDB
   - Enter the **Flush interval** and select the **Unit** (Please note that too short interval might cause data loss!)

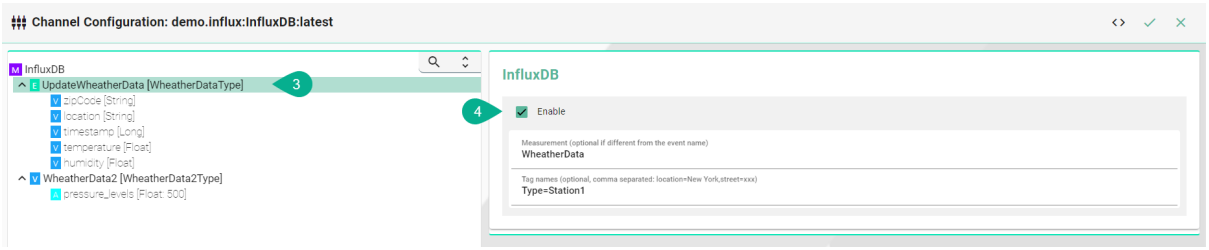


Description of configuration properties

Property	Description	Example
URL	Database URL and port	`http://127.0.0.1:8086`
DB Name	Database name	`InfluxDB`
Credentials	Database credentials	`None`
Batch size	Data written in batches	`1000`
Flush interval	Delay between data flushes in milliseconds, at most batch size records are sent during flush	`1000`
Measurement	Name of the measurement stored in influxdb	`WeatherData`
Tag names	Optional tag to be added to the measurement	`Type=Station`

Writing / Event Configuration

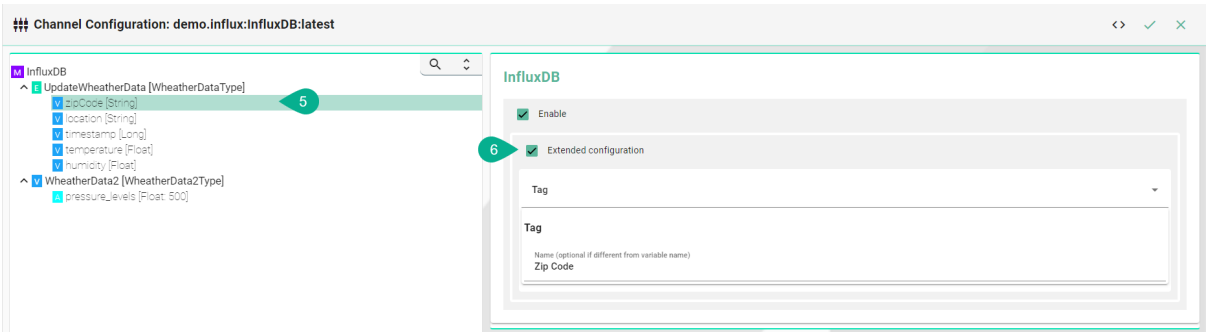
3. Select the **event node**
4. Enable the checkbox to configure the event
    - Enter the **Measurement** - if it differs from the event name
    - Enter **Tags** - comma separated



Tags

Tags are metadata for the data. They're made up of key-value pairs, and they describe attributes of the data that don't change every time the data point is recorded. To configure a variable as a **tag** follow the steps below:

5. Select the variable which should be a **Tag**
6. Enable **Extended configuration**
    - Select **Tag** from the drop-down menu
    - Enter a **Name** - if it differs from the variable name

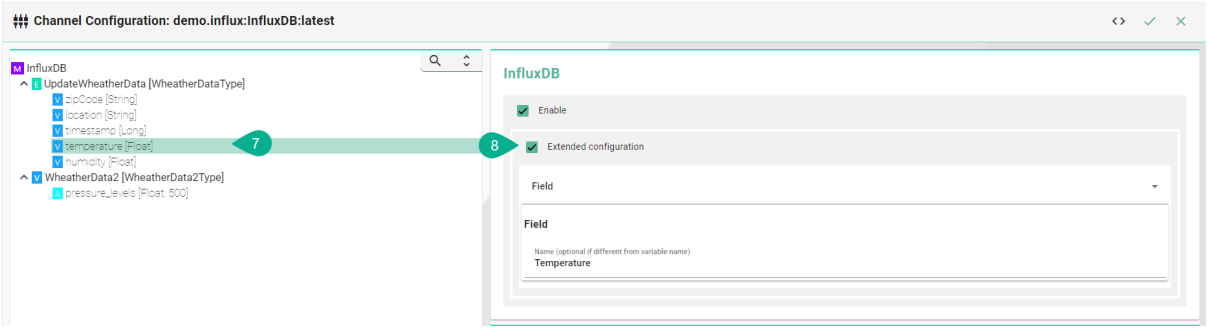


Fields

Fields represent the actual data stored and consist of key-value pairs. Unlike tags, fields aren't indexed. Variables not explicitly configured are automatically recognized as fields by the InfluxDB Channel.

If the field name should differ from the variable name in the Information Model, follow the steps below:

7. Select the variable which should be a **field**
8. Enable **Extended configuration**
    - Select **Field** from the drop-down menu
    - Enter a **Name** - if it differs from the variable name



Time

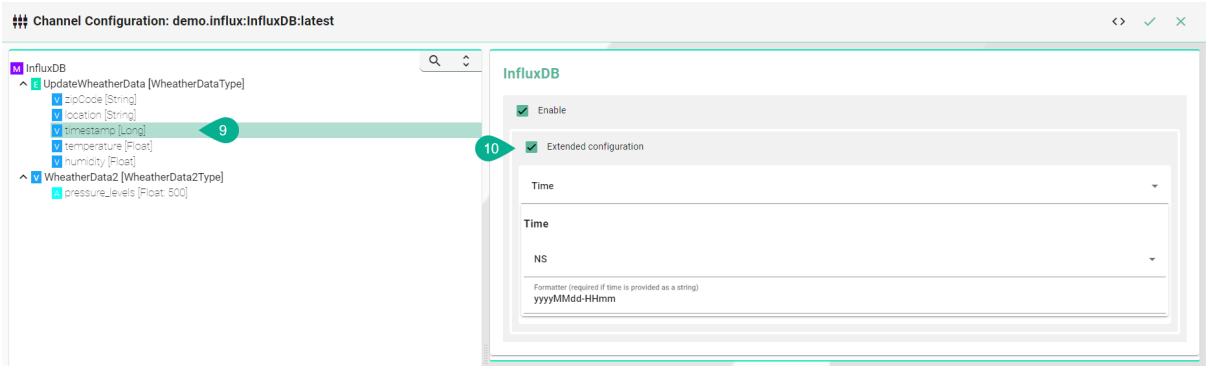
Timestamps indicates when a data point occurred and, in combination with its tag set, uniquely identifies that data point in a series. The time can be provided as:

- Long value (unix timestamp)

- String (format needs to be provided in the configuration)
- OffsetDateTime

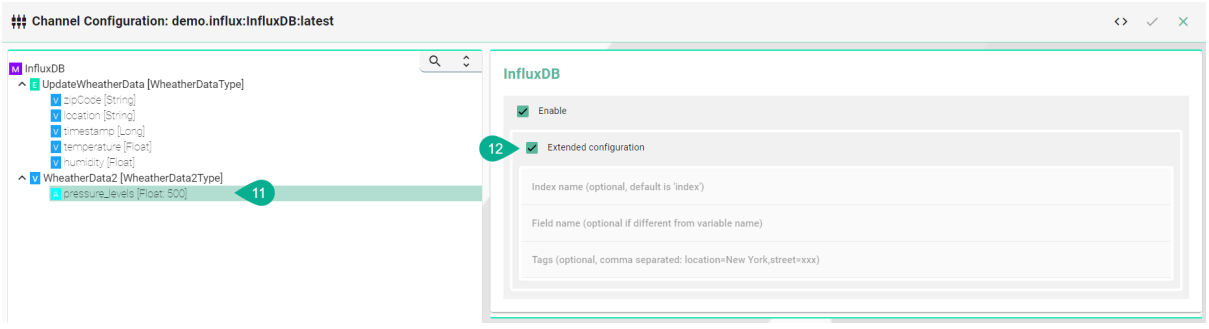
To configure a variable as a **time** follow the steps below:

9. Select the variable which should be a **time**
10. Enable **Extended configuration**
    - Select **Time** from the drop-down menu
    - Select the **Precision** (Only for variables of type *Int* or *Long*)
    - Enter a **Formatter** (This is required if time is provided as a *String*)



Arrays

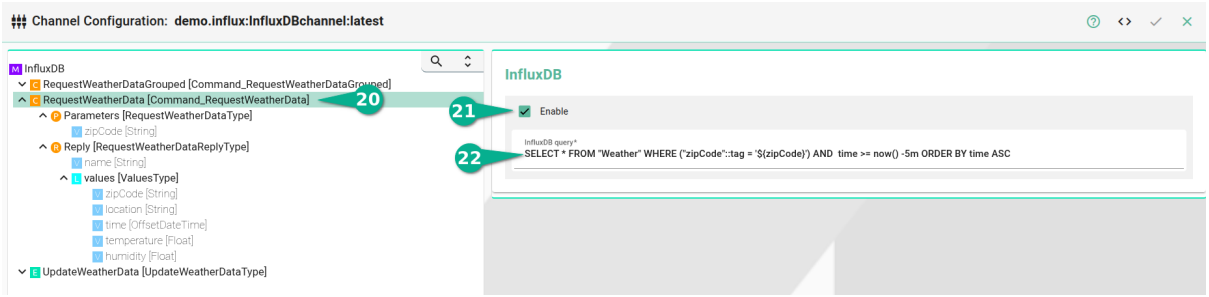
11. Select the Array
12. To configure the Array select **Extended Configuration**
    - (Optional) Enter an **Index** name
    - (Optional) Enter a **Field** name if the event node name differs from the actual name in InfluxDB.
    - (Optional) Enter **Tags** separated by commas e.g., (location=NewYork, street=xxx)



Reading / Command Configuration

20. Select the **command node**
21. Enable the checkbox to configure the command
22. Enter the query.

    For using variables in the query the `${MyVariableName}` pattern can be used



InfluxDB v2

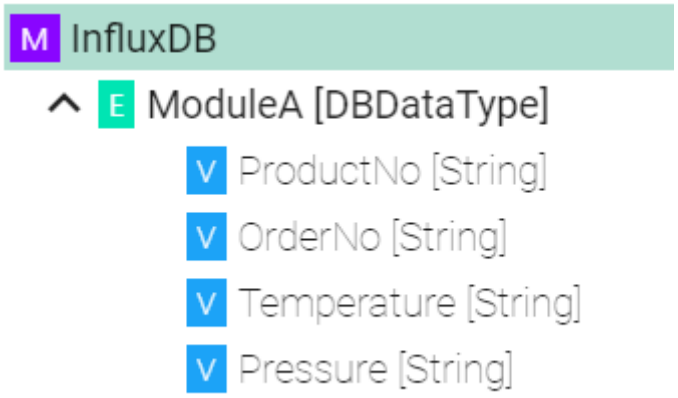
Characteristics - InfluxDB v2

In case of a time series data use case where you need to ingest data in a fast and efficient way you can use InfluxDB.

Information Model Requirements

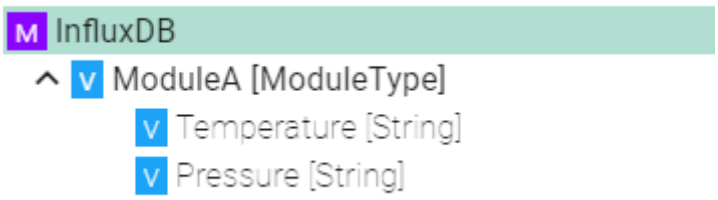
Inserts using Events

- The node after the root model in this case is of the type *Event* E which represent a database table.
- Fields are represented by *Variables* V.



Inserts using Custom Data Types

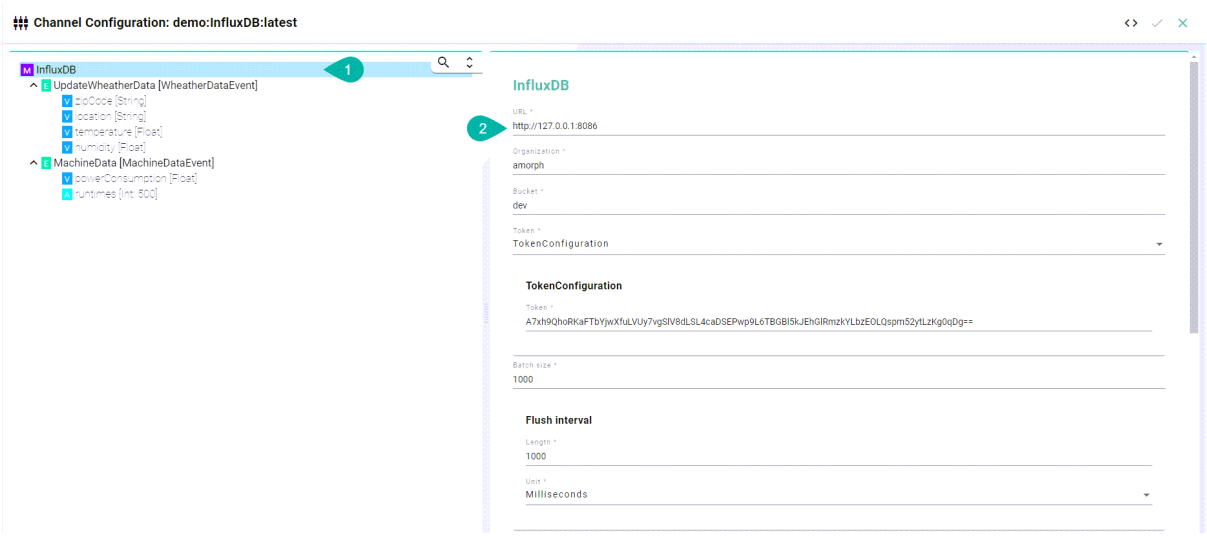
- Complex *Variables* V (ModuleA) represents Measurements
- *Variables* V underneath within the complex variable (Temperature) represents Fields



- Arrays A can be used to set use an index

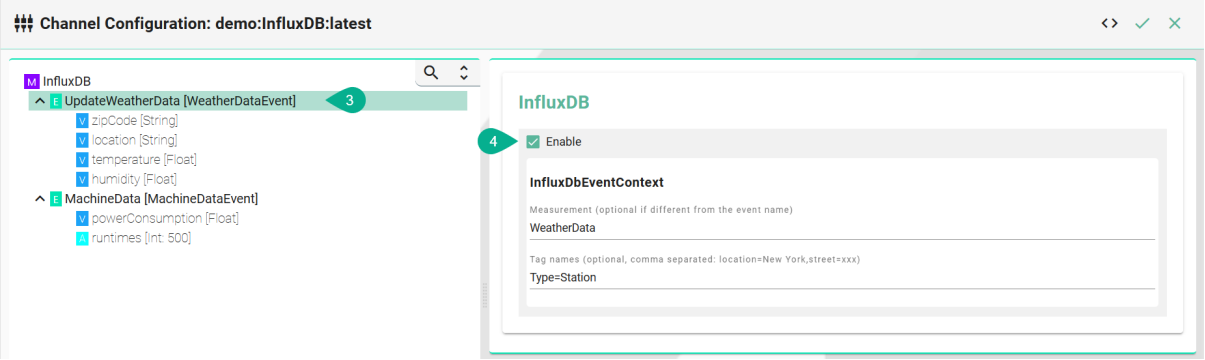
How to configure InfluxDB v2

1. Select the **root model node** in the tree on the left.
2. Configure the InfluxDB.
    - Enter the **URL** to the database
    - Enter the **Organization** defined in the database
    - Enter the **Bucket** defined in the database
    - Enter the **Token** or select it from the Credential Manager
    - Enter the **Batch size** - writes data in batches to minimize network overhead when writing data to InfluxDB
    - Enter the **Flush interval** and select the **Unit** (Please note that too short intervalls might cause data loss!)



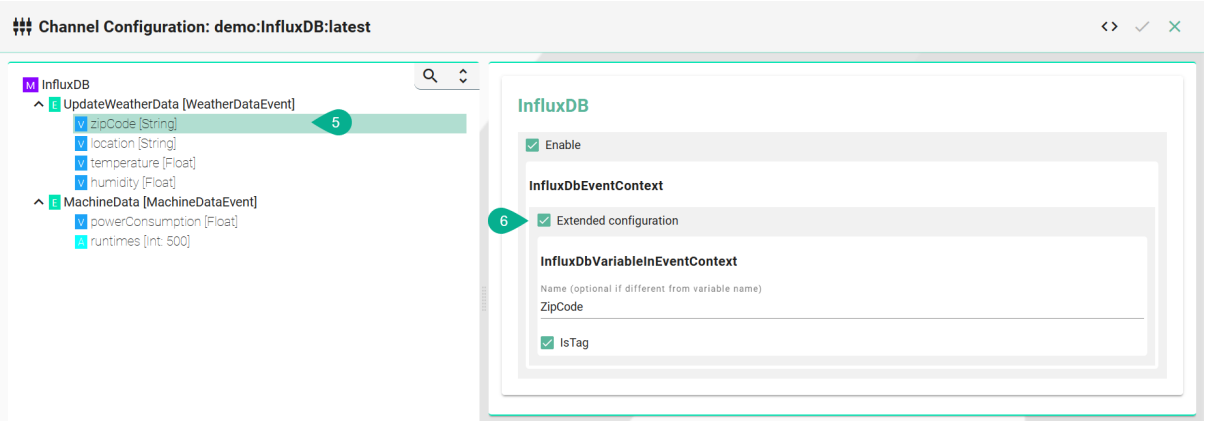
Event Configuration

1. Select the **event node**
2. Enable the checkbox to configure the event
    - Enter the **Measurement** - if it differs from the event name
    - Enter **Tags** - comma separated



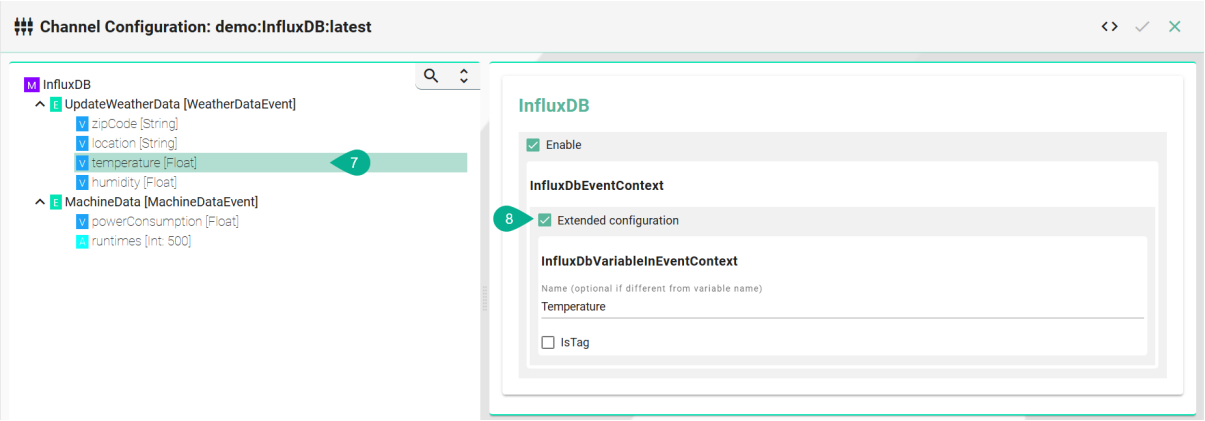
Configuration of Tags

5. Select the variable which should be a **Tag**
6. Enable **Extended configuration**
    - Enter a **Name** - if it differs from the variable name
    - Enable the checkbox **IsTag**



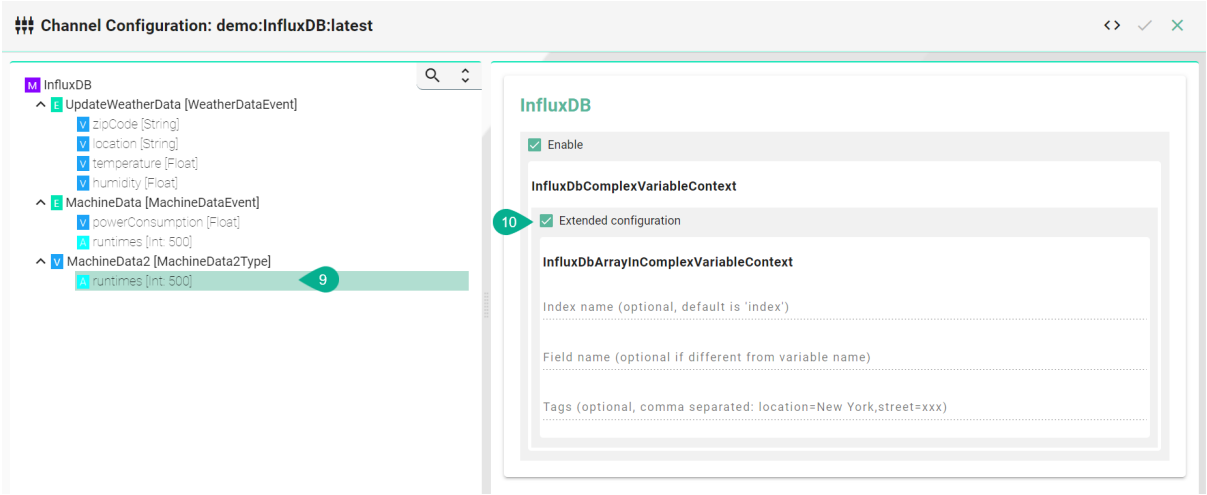
Configuration of fields

7. Select the variable which should be a **field**
8. Enable **Extended configuration**
    - Enter a **Name** - if it differs from the variable name
    - Leave the checkbox **IsTag** disabled



Array Configuration

9. Select the Array
10. To configure the Array select **Extended Configuration**
    - (Optional) Enter an **Index** name
    - (Optional) Enter a **Field** name if the event node name differs from the actual name in InfluxDB.
    - (Optional) Enter **Tags** separated by commas e.g., (location=NewYork, street=xxx)



Description of configuration properties:

Property	Description	Example
URL	Database URL and port	`http://127.0.0.1:8086`
Organization	Name of the Organization	`CompanyName`
Bucket	Name of the Bucket	`Database_1`
Credentials	Token-based authentication	Token
Batch size	Data written in batches	`1000`
Flush interval	Delay between data flushes in milliseconds, at most batch size records are sent during flush	`1000`
Measurement	Name of the measurement stored in influxdb	`WeatherData`
Tag names	Optional tag to be added to the measurement	`Type=Station`

InMemory

Characteristics - InMemory

The InMemory Communication Channel can be used to cache data for reuse in the Mapping.

Data can be stored persistently by writing the data structure of the Model and the values of the Variables to a file in **JSON format**. If no values are mapped to the Variables of the InMemory

Information Models, the values are set to *null* an initial value.

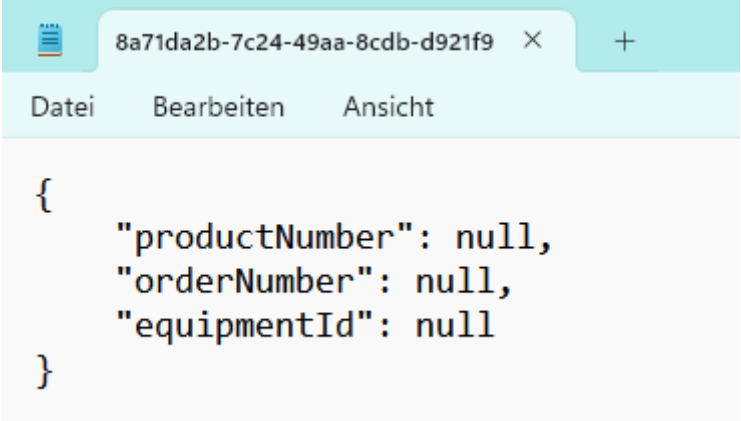


Fig. 1: Example JSON file

Information Model Requirements

The following Node Types can be used to model data structures:

- *Variables* with a *Simple Data Type*.
- *Variables* with a *Custom Data Type*.

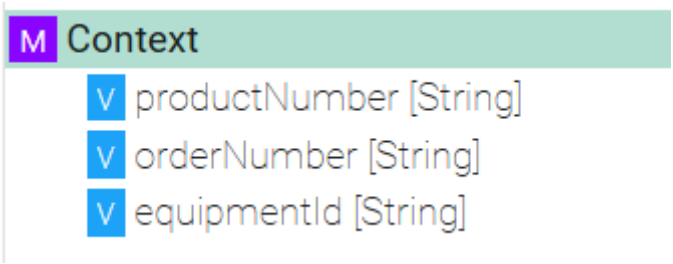
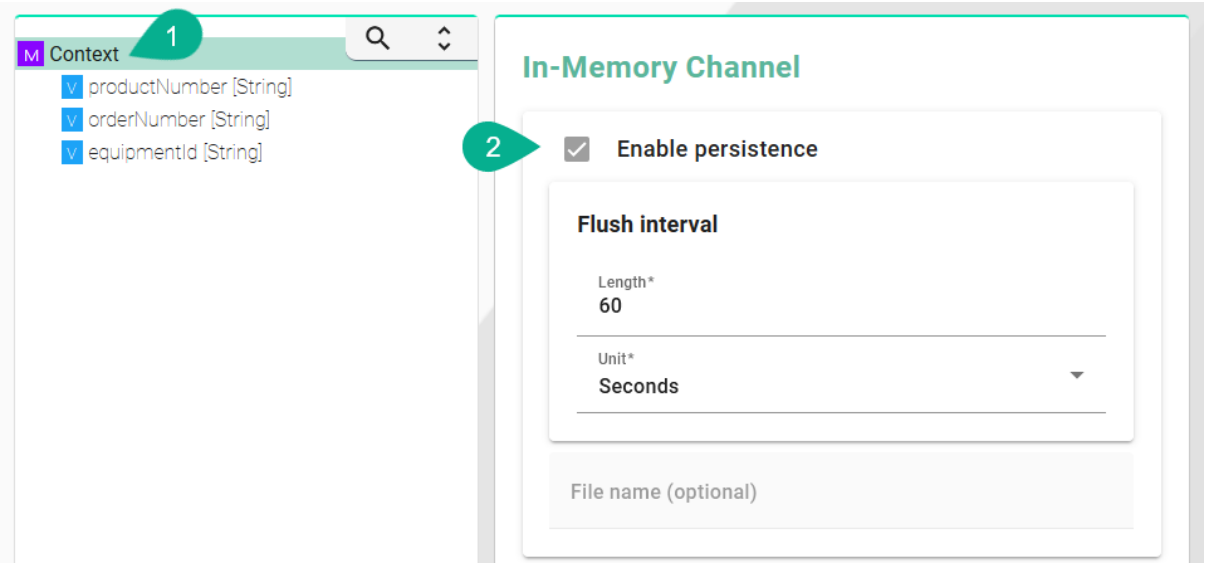


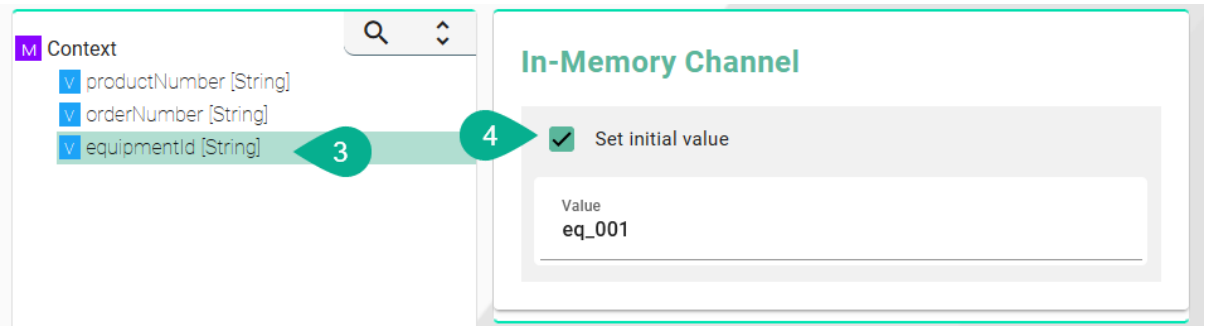
Fig. 2: Example Information Model

How to configure InMemory

1. Select the **root model node**
2. (Optional) Enable **persistence** if the data should be stored persistent
    - Adjust if necessary the **Flush interval** — it determines how often data should be persisted (flushed) to a file
    - Enter a **File name** - by default the identifier (Id) of the Communication Channel is set as file name



3. Select a **Variable** if an initial value should be set
4. Enable **Set initial value** and enter some data



Protocols

MQTT Client

Characteristics

MQTT is an OASIS standard messaging protocol for the Internet of Things (IoT). To learn more about the standard visit the MQTT website.

Information Model Requirements

- The first node after the root node, M , must be of type *Event* E or *Variables* V with a *Simple Data Type* or *Complex Date Type*.
- Properties can be used *Properties* P for dynamic topics within Events and Complex Variables
- The following Node Types can be used under the Event Node:
  - *Variables* V with a *Simple Data Type* represents the key-value pairs.
  - *Variables* V with a *Custom Data Type* represent objects that can contain key-value pairs.

- With *Lists* L you can aggregate multiple variables.

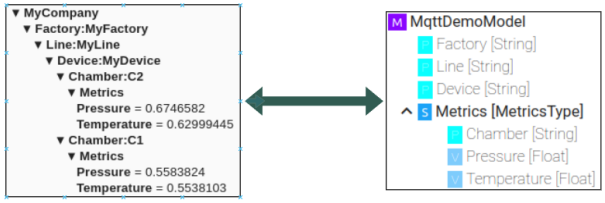
Hint

When publishing a topic, the Information Model dictates the payload's structure.

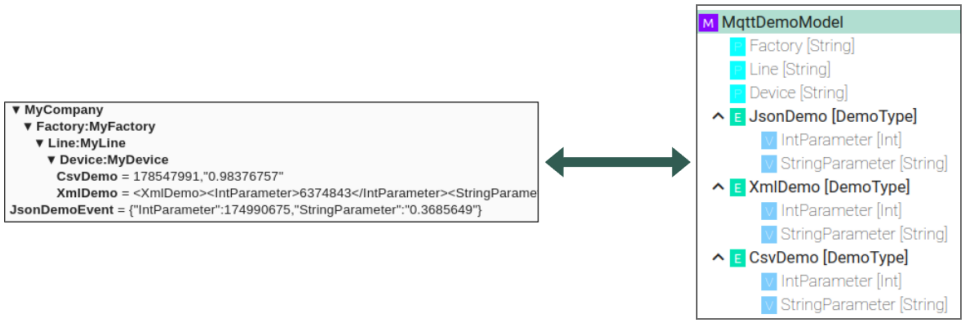
Hint

When subscribing to a topic, ensure that the Information Model structure aligns with the payload.

Complex Variables nodes



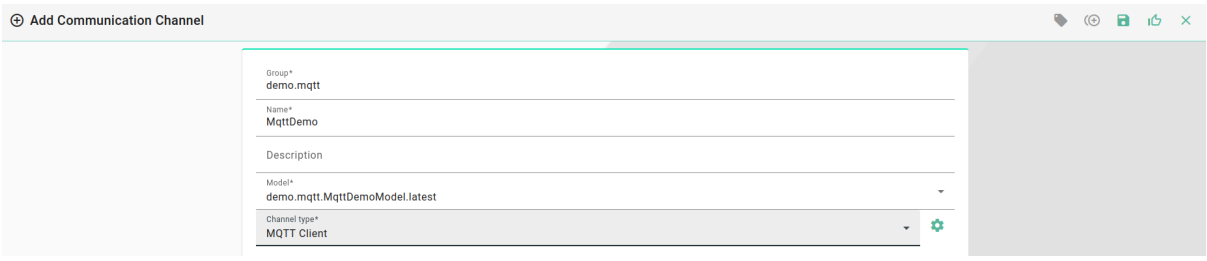
Event nodes



Configuration

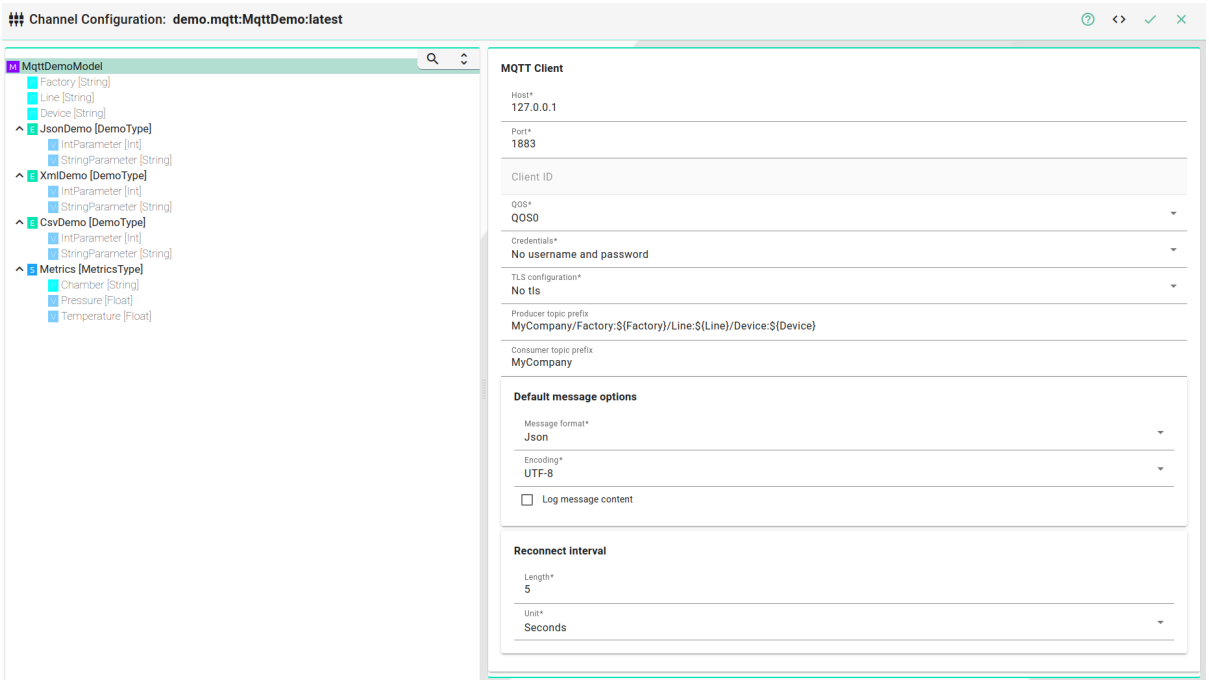
Model

1. Select the **MQTT Client** as Channel Type.
2. Click the **Configure** button.



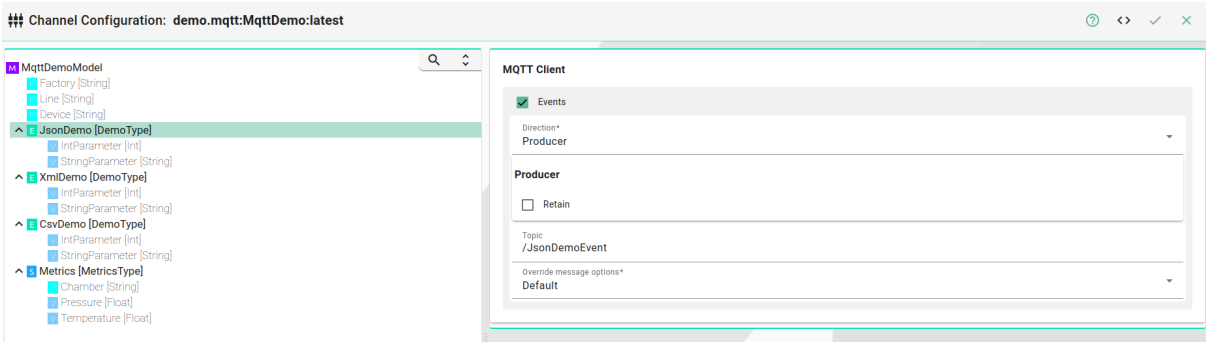
3. Select the **root model node**
4. Configure the MQTT client:

- Enter **Host** and **Port** of the MQTT Broker used
- (Optional) Specify a **Client ID**
- Set the **Quality of Service (QoS)**
- If required configure the credentials.
- If required configure TLS
  - (Optional) Provide CA certificate to validate the host
  - (Optional) Provide Client certificate and key file for client authentification
- (Optional) Provide a prefix for the topic that will be used when publishing messages
- If required, adjust the default values for **Reconnect interval** and the **Unit**
- (Optional) Change the default message options. These can be overwritten for each event



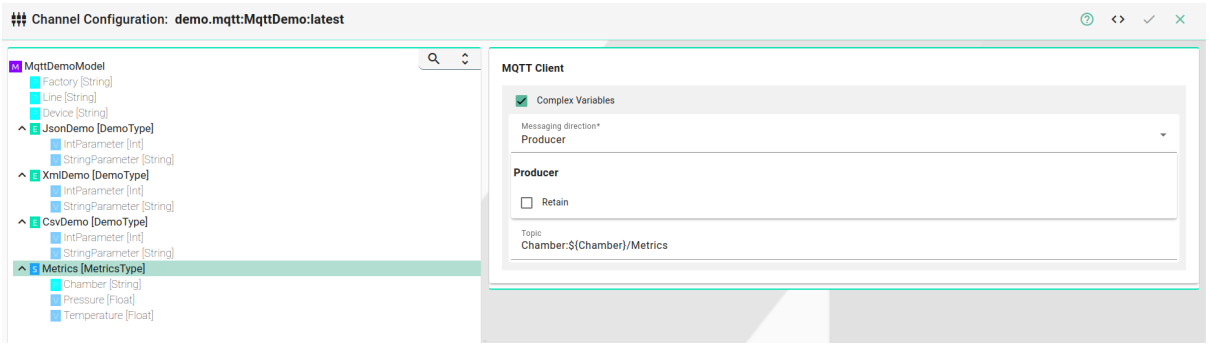
Events

1. Select the **event node** in the tree on the left.
   - Select the **Direction** (Producer or Consumer)
   - For Producer set **Retain** if required
2. Configure the topic.
3. Optional override the default message options



Variables

1. Select the **variable node** in the tree on the left.
    - Select the **Direction** (Producer or Consumer)
    - For Producer set **Retain** if required
2. Configure the topic.
3. Optional override the default message options



Topics

The topics for publishing and subscribing can be configured in a static but also dynamic why where variables and properties are used within the topic

Producer

- In the Model configuration a prefix (`Producer topic prefix`) can be configured.
- This prefix will be automatically added to the topics for producer events and variables.
- If variables/properties are going to be used they need to be written in the `${VariableName}` pattern.
- In the producer prefix only variables that are directly in the root node of the Model can be used.
- In the topic configuration of the events / complex variables only variable and properties with the complex variable can be used
- If the topic in the event / complex variable configuration starts with a "/" then the prefix will not be added

Example

Following variable values are used in the example:

Name	Value
Factory	MyFactory
Line	MyLine
Device	MyDevice
Chamber	15

Topic configuration

Producer prefix	Event / complex variable topic	Topic (Example)
`MyCompanyNamespace`	Complex variable: `Metrics`	`MyCompanyNamespace/Metrics`
`Factory:${Factory}/ Line:${Line}/ Device:${Device}`	Complex variable: `Chamber:${Chamber}/ Metrics`	`Factory:MyFactory/Line:MyLine/ Device:MyDevice/Chamber:15/ Metrics`
`Factory:${Factory}/ Line:${Line}/ Device:${Device}`	Complex variable: `/ Chamber:${Chamber}/ Metrics`	`/Chamber:15/Metrics`

Consumer

- In the Model configuration a prefix (`Consumer topic prefix`) can be configured.
- This prefix will be automatically added to the topics for consumer events and variables.
- Variables/properties can be used as wildcards in events and complex variables. They need to be written in the `${VariableName}` pattern.
- Wildcards are not supported in the prefix
- Wildcards variables/properties can only be used for an entire topic segment. For example `MyNamespace/${Factory}/Metrics` which will be translated into `MyNamespace/+/ Metrics`
- A partial topic segment is not possible. For example `MyNamespace/Factory:${Factory}/ Metrics`
- If the topic in the event / complex variable configuration starts with a "/" then the prefix will not be added

Example

Consumer prefix	Event / complex variable topic	Topic (Example)
`MyCompanyNam`	Complex variable: `Metrics`	`MyCompanyNamespace/ Metrics`
`MyCompanyNam`	Complex variable: `${Factory}/${Line}/ ${Device}/${Chamber}/Metrics`	`MyCompanyNamespaceFactory/ +/+/+/+/Metrics`

MQTT (deprecated)

This mqtt client is deprecated and will be removed in 1.11. Please use *MQTT Client* for new implementations.

Characteristics - MQTT

MQTT is an OASIS standard messaging protocol for the Internet of Things (IoT). To learn more about the standard visit the MQTT website.

Information Model Requirements

- The first node after the root node, M, must be of type *Event* E or *Variables* V with a *Simple Data Type*.
- The following Node Types can be used under the Event Node:
  - *Variables* V with a *Simple Data Type* represents the key-value pairs.
  - *Variables* V with a *Custom Data Type* represent objects that can contain key-value pairs.
  - With *Lists* L you can aggregate multiple variables.

Hint

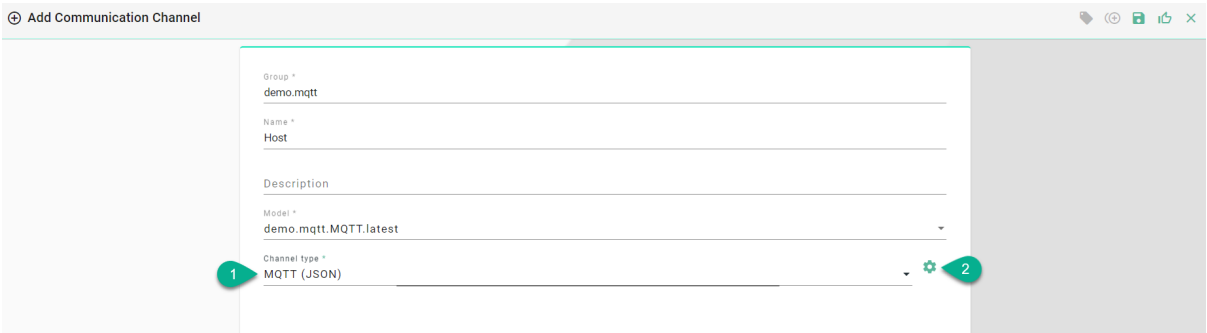
When publishing a topic, the Information Model dictates the payload's structure.

Hint

When subscribing to a topic, ensure that the Information Model structure aligns with the payload.

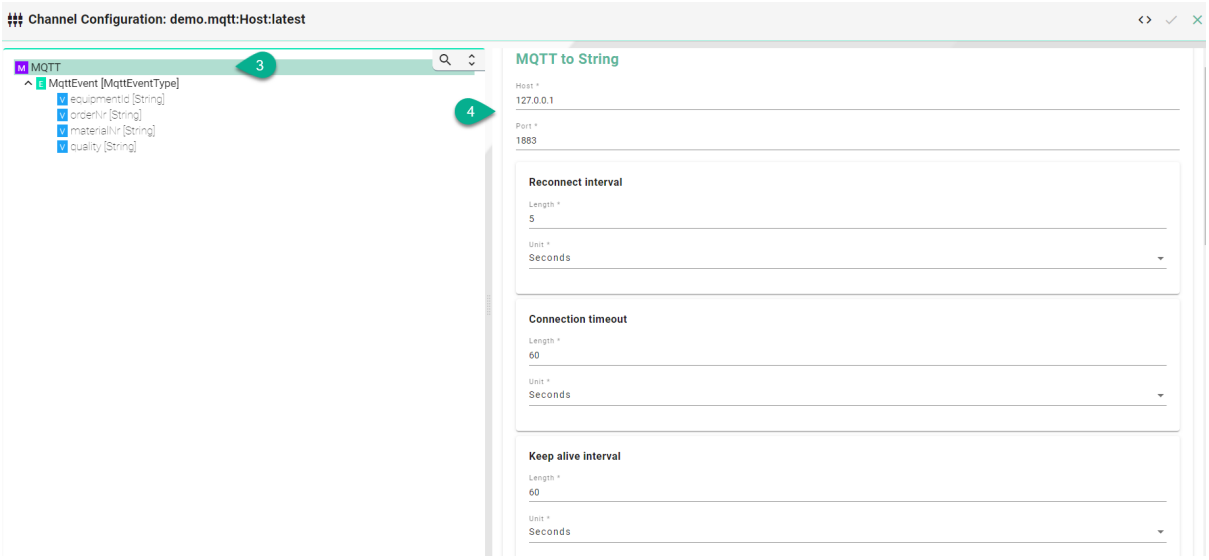
Configuration - MQTT Channel

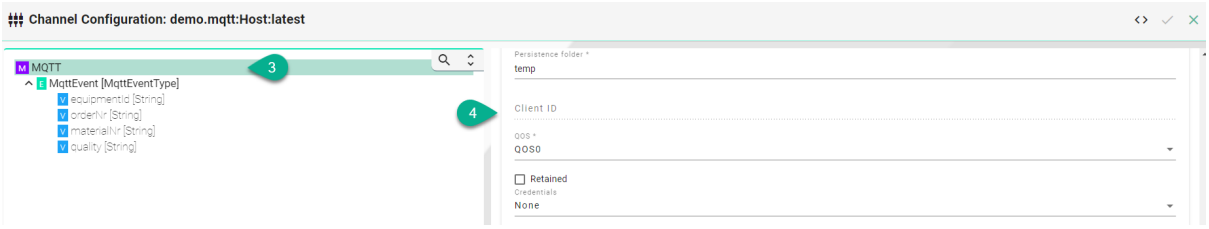
1. Select the **MQTT (JSON)** as Channel Type.
2. Click the **Configure** button.



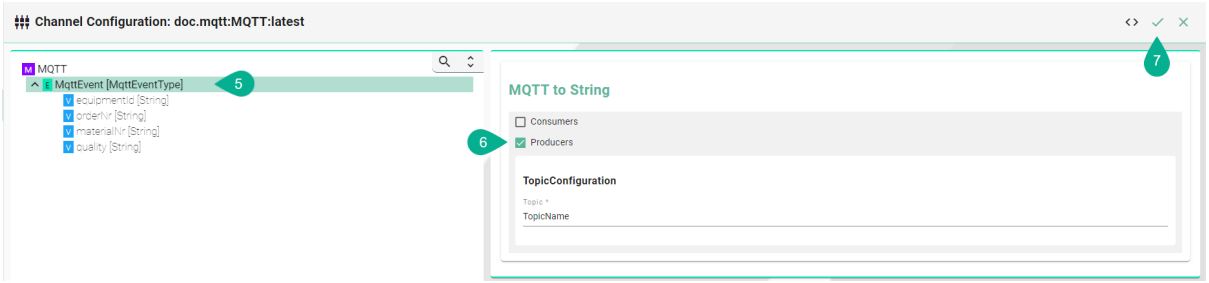
3. Select the **root model node**
4. Configure the MQTT To String configuration:

- Enter **Host** and **Port** of the MQTT Broker used
- If required, adjust the default values for **Reconnect interval**, **Connection timeout**, **Keep alive interval** and the **Unit** for each
- Specify a path to a folder on your local machine. The **temp** directory inside the *SMARTUNIFIER Manager* can be used as well.
- (Optional) Specify a **Client ID**
- Set the **Quality of Service (QoS)**
- (Optional) Enable **Retained** if required
- Select **Username and password** in order to manually enter the credentials or select **Username and password credentials reference** to add it from the Credentials Manager. If there are no credentials needed (e.g., `test.mosquitto.org`) select **None**.



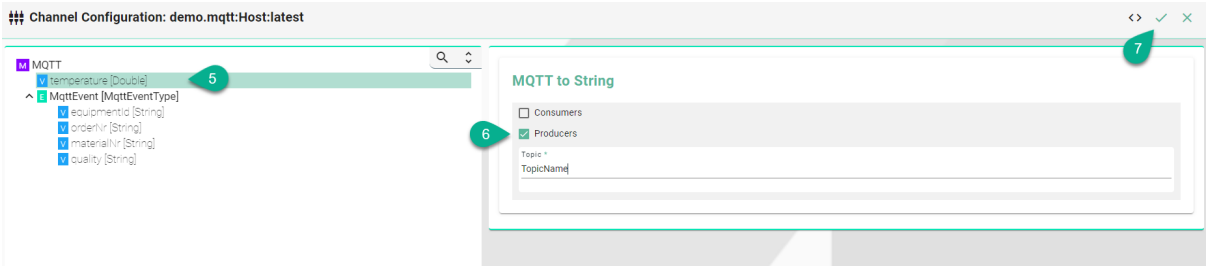


5. Select the **event node** in the tree on the left.
6. Enable either **Producer** or **Consumer** depending on the use case and enter a **Topic name**.
7. Click the **Apply** button.



Note

The Producer and Consumer options can be enabled for a Variable node, allowing it to produce or subscribe to/from a single variable

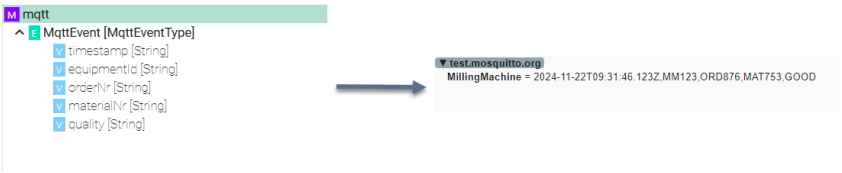


Following are some examples of different data types:

CSV

- CSV columns are represented by the Node Type *Variables* . Note that the order of the Variables in the Information Model determines the order of the columns in the MQTT output.

Here is an example of the Information Model and how an CSV input is displayed in MQTT:



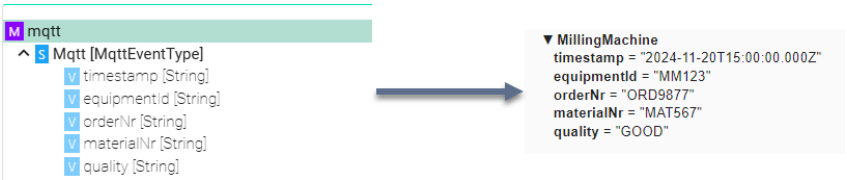
JSON

Here are some examples, of how a JSON could be output in MQTT:

When dealing with complex variables, it is possible to produce to multiple subtopics by configuring them individually.

For example:

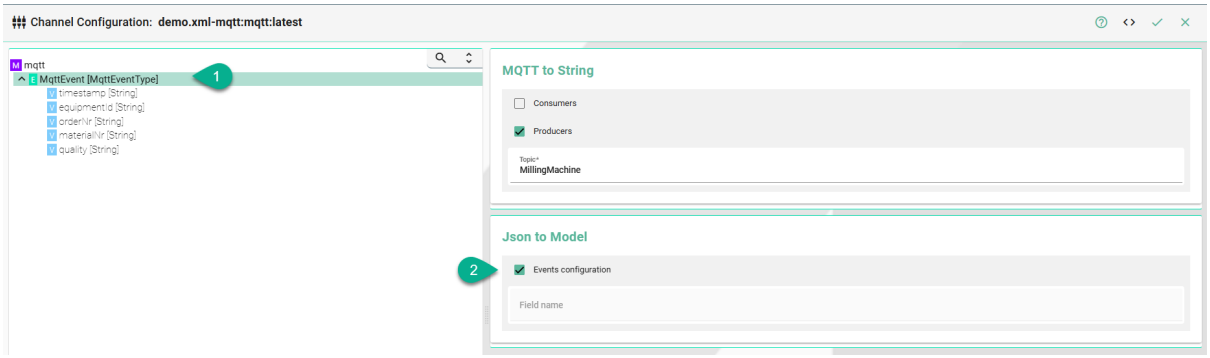
- The **complex variable** "Mqtt" has the default topic "Mqtt", this can be changed if needed.
- The **nodes** within that topic are structured as "Mqtt/nodename", when outputted in mqtt.



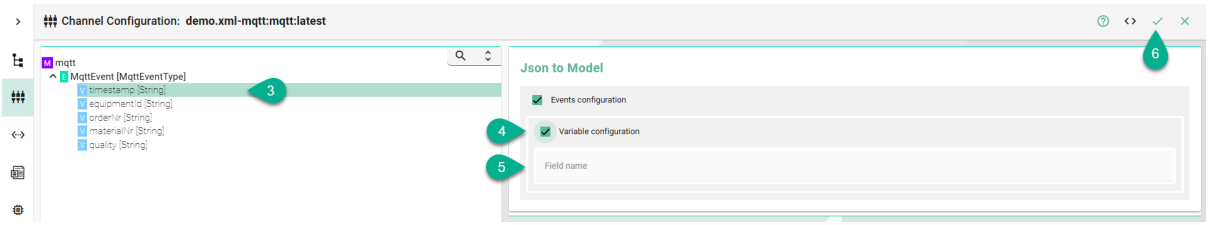
Variable Configuration

This configuration can be used when certain keywords or reserved words from the data set are not allowed in the Information Model. For example, for the *Scala keywords* like 'type', you might name it 'Type' in the Information Model. Subsequently, in the Channel configuration, you can assign its actual name using the **Field name** input.

To configure the **field name** follow the steps described below:



1. Select the **event** in the tree on the left.
2. Check the box for the **Events configuration**.
3. Select the **variable** in the tree on the left.
4. Check the box for the **Variable configuration**.
5. Input the **Field name**, representing the reserved word.
6. Click on the **Apply** button.



Certificates

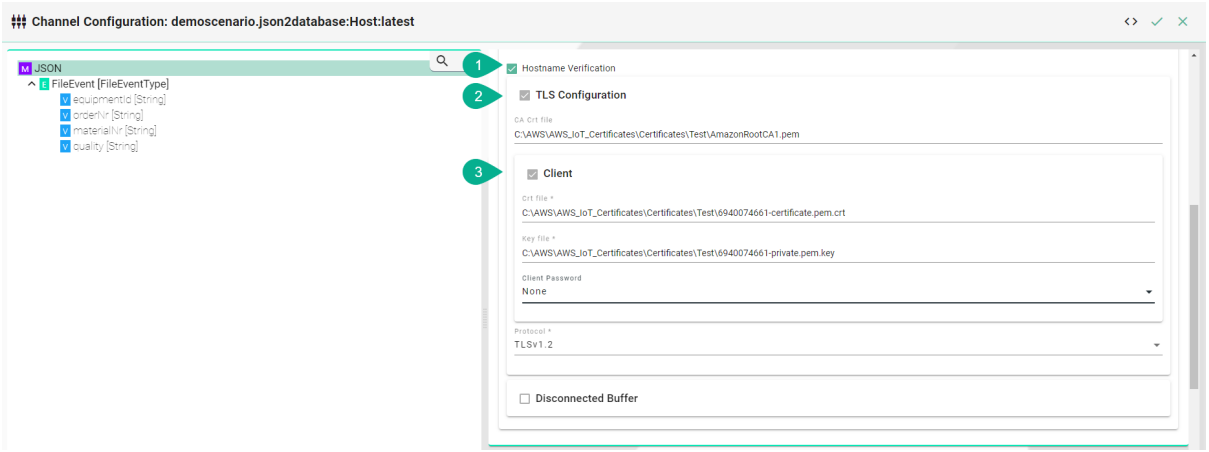
Encrypted connection using TLS security is supported. Follow the steps below to encrypt the connection.

1. Enable **Hostname Verification** (optional)
2. Enable the **Tls Configuration** checkbox
   - Enter the **path** to the **CA (certificate authority) certificate** of the CA that has signed the server certificate

> **Note**
>
> Make sure the CA certificate is valid.

3. Enable the **Client** checkbox
   - Enter the **path** to the **Client certificate**. The client certificate identifies the client just like the server certificate identifies the server.
   - Enter the **path** to the **Private client key**.

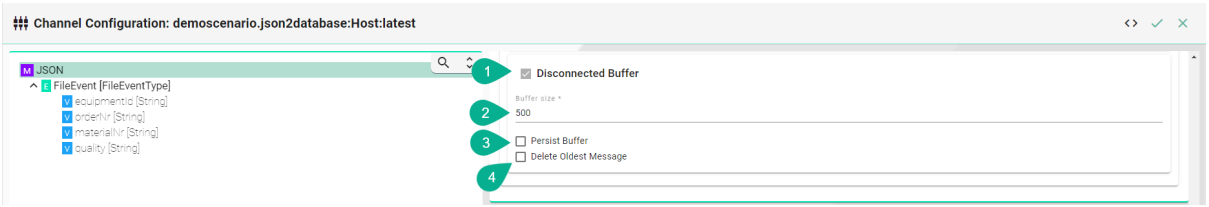
- If applicable select to enter a **Password** or to add from the **Credentials Manager**.
- Select the **Protocol** from the Drop-Down.



Disconnected Buffer

In case the connection is lost, messages can be buffered offline when the Disconnected Buffer is enabled. Follow the steps below to enable the DisconnectedBuffer.

1. Enable the **Disconnected Buffer** checkbox.
2. Set the **Buffer size** - defines the number of messages being hold e.g., 5000.
3. (Optional) Enable **Persist Buffer**.
4. (Optional) Enable **Delete Oldest Message**.



Description of configuration properties:

Property	Description	Example
host	URL of the MQTT Broker.	`test.mosquitto.org`
port	Port of the MQTT Broker.	`1883`
reconnectInterval	Time interval to reconnect to the MQTT Broker after loss of connection in seconds	`5`
connectionTimeout	Time interval the connection times out in seconds	`60`
keepAliveInterval	Time the session persists in seconds	`60`
persistenceFolder	Path to a folder for the persistence store of the MQTT	`temp`
clientId	Identifies an MQTT client which connects to an MQTT Broker	`MyClientID`
username	Client username	`Username`
password	Client password	`Password`
hostnameVerification	Hostname Verification	`true`, `false`
tls	Encryption	`true`, `false`
producers	Data producer	`true`, `false`
consumer	Data consumer	`true`, `false`
protocol	TLS protocol version	`TLSv1.1`, `TLSv1.2`
disconnectedBuffer	Offline buffering of data	`true`, `false`
bufferSize	Amount of message allowed in the buffer	`5000`
persistBuffer	Buffer persistence	`true`, `false`
deleteOldestMessage	Delete oldest message in buffer	`true`, `false`

Modbus TCP/IP

Characteristics

MODBUS is an application-layer messaging protocol, positioned at level 7 of the OSI model. It provides client/server communication between devices connected on different types of buses or networks. To learn more about the standard visit the MODBUS website.

Supported features

- FC03 Read Holding Registers
- FC04 Read Input Registers

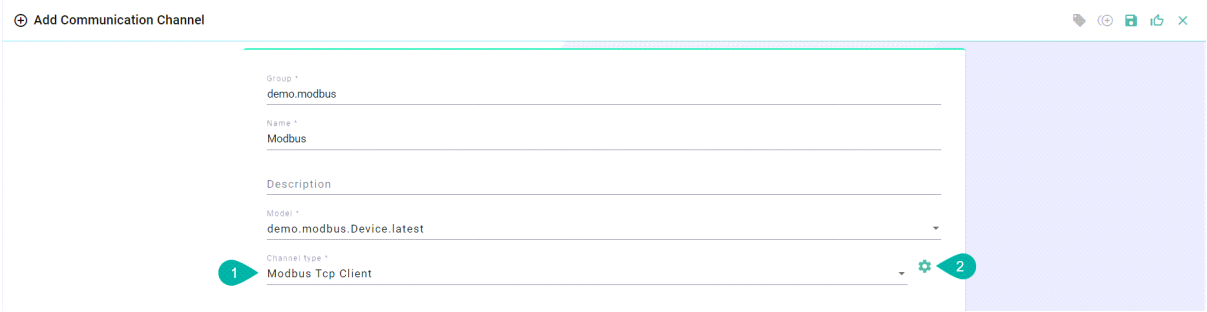
Information Model Requirements

The following Node Types can be used to model a register:

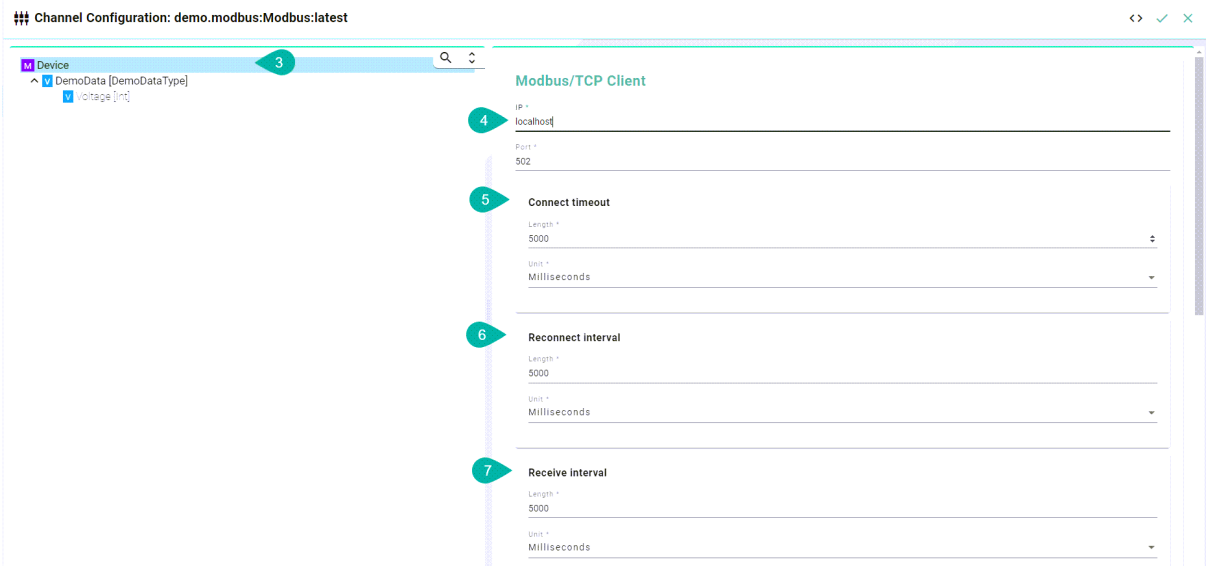
- *Variables* with a *Custom Data Type* containing *Variables* with a *Simple Data Type*.

Configuration

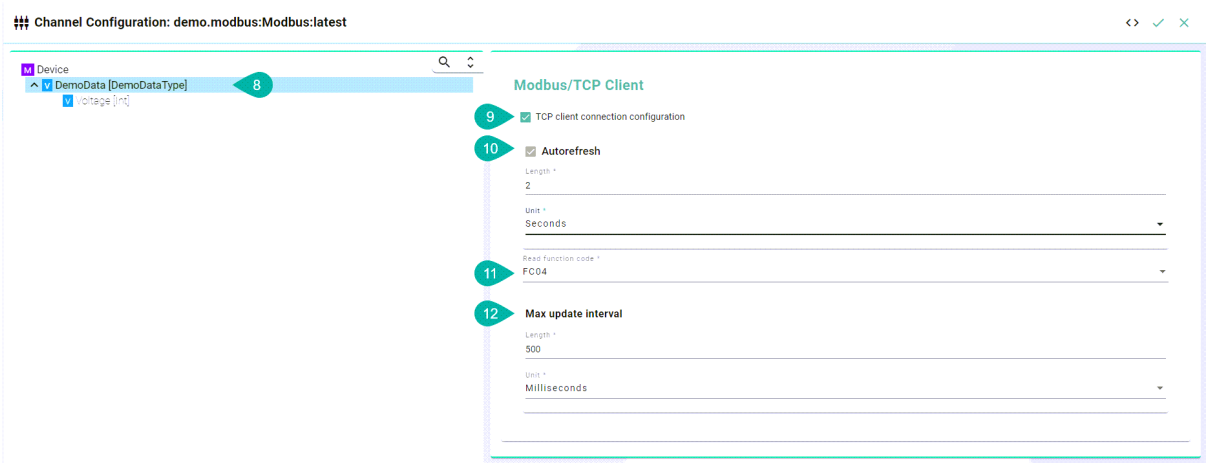
1. Select **Modbus/Tcp Client** as Channel Type.
2. Click the **Configure** button.



3. Make sure the root model node is selected to configure the Modbus/TCP Client
4. Enter the **IP** address and the **port**
5. (Optional) Change the **Connect interval** if needed
6. (Optional) Change the **Reconnect interval** if needed
7. (Optional) Change the **Receive interval** if needed

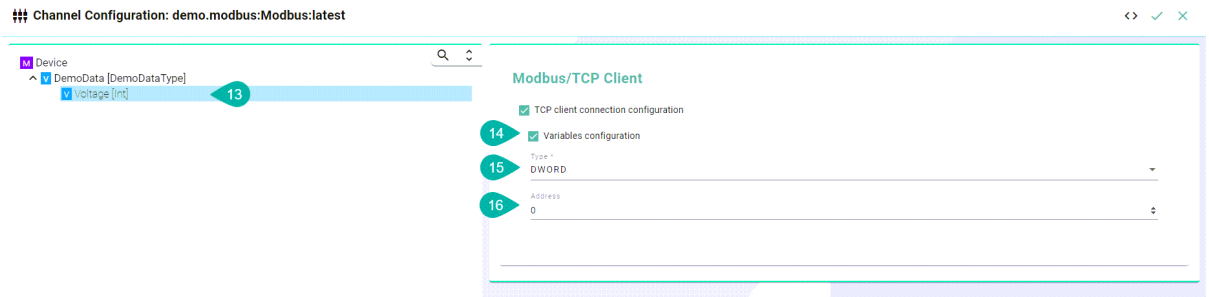


8. Select the complex variable node
9. Enable the checkbox **TCP Client connection configuration**
10. (Optional) Enable **Autorefresh** to specify the retrieval rate
11. Select the **Function Code**
12. (Optional) Change the **Max update interval** if needed



13. Select the complex variable node
14. Enable the checkbox **Variables configuration**
15. Select the **Data type**
16. (Optional) Enter the **register address**

> **Note**
>
> If **address** is left empty, SMART**UNIFIER** assumes that the Information Model structure is in line with the register addresses.



Data type formats

Data Type	Size	Range
BYTE, USINT, UInt8	8 Bit	0 - 255
WORD, UINT, UInt16	16 Bit	0 - 65.535
DWORD,UDINT, UInt32	32 Bit	0 - 4.294.967.295
LWORD,ULINT, UInt64	64 Bit	0 - 2^64-1
SINT, Int8	8 Bit	-128 - 127
INT, Int16	16 Bit	-32.768 - 32.767
DINT, Int32	32 Bit	-2.147.483.648 - 2.147.483.647
LINT, Int64	64 Bit	-2^63 - 2^63-1
REAL, Float32	32 Bit	-3,402823e+38 - 3,402823e+38
LREAL, Float64	64 Bit	-1,7976931348623158e+308 - 1, 7976931348623158e+308

Description of configuration properties

Property	Description	Example
IP	Client IP	`localhost`
Port	Client port	`502`
Connection timeout	Time interval the connection times out	`60`
Reconnect interval	Time interval to reconnect to the client after loss of connection	`5`
Receive interval	TCP/IP receive timeout	`50`
Autorefresh	Automatic polling of Modbus server	`2`
Read function code	Function code used for reading variables from a modbus server	`FC04`
Max update interval	Minimum time between requests to the Modbus server (if autorefresh is not used)	`60`
Variable configuration Type	Format of variable	`DWORD`
Variable configuration Address	Address of the variable on the modbus server	`0`

OPC-UA Client

Characteristics

OPC (Open Platform Communications) enables access to machines, devices and other systems in a standardized way. To learn more about the standard visit the OPC-UA website.

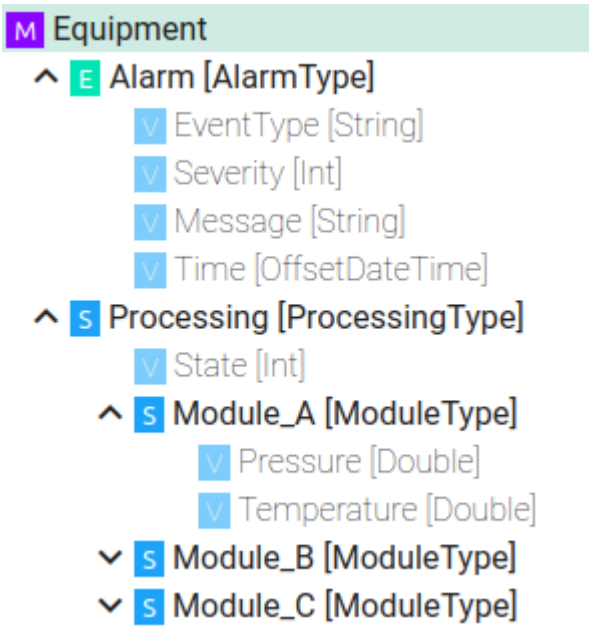
Supported features

- Data access
- Events (e.g. Alarms)
- Authentification

Information Model Requirements

The following Node Types can be used to model data structures:

- *Event* E
- *Variables* V with a *Simple Data Type*.
- *Variables* V with a *Custom Data Type*.



Configuration

Connection settings

OPC-UA Client

OPC UA client application name*
demo

OPC UA client application URI (auto-generated if not set)

OPC UA server TCP connection settings

OPC UA server IP address or hostname*
192.168.138.71

OPC UA server TCP port*
4840

OPC UA server endpoint path (appended to opc.tcp://host:port/)
/

Client authentication method (Anonymous or UsernamePassword)*
No authentication (anonymous access)

OPC UA security policy for the secure channel*
No security

OPC UA message security mode (Auto, None, Sign, or SignAndEncrypt)*
Automatic selection: prefer SignAndEncrypt > Sign > None

Request timeout in milliseconds*
5000

Interval between reconnection attempts

Length*
3

Unit*
Seconds

Maximum number of connection retries (0 = unlimited)
0

Client certificate keystore for OPC UA secure channel*
No client certificate keystore

Display format for OPC UA NodeIds*
Node identifier as a human-readable name string

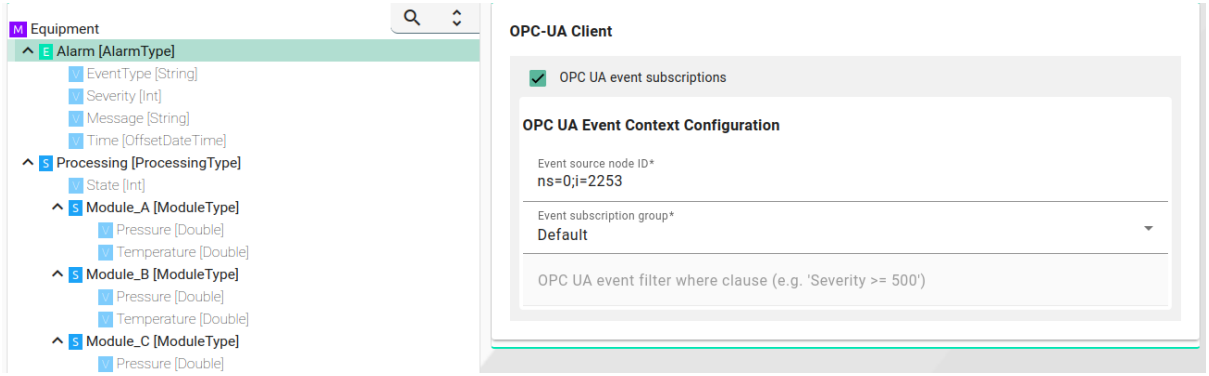
Property	Description	Example
OPC UA client application name	The name of the OPC UA client application.	demo
OPC UA client application URI	The URI of the OPC UA client application (auto-generated if not set).	(auto-generated)
OPC UA server IP address or hostname	The IP address or hostname of the OPC UA server.	192.168.138.71
OPC UA server TCP port	The TCP port of the OPC UA server.	4840
OPC UA server endpoint path	The endpoint path appended to *opc.tcp://<host>:<port>/*.	/
Client authentication method	The authentication method (Anonymous or UsernamePassword).	No authentication (anonymous access)
OPC UA security policy for the secure channel	The security policy applied to the secure channel.	No security
OPC UA message security mode	The message security mode (Auto, None, Sign, or SignAndEncrypt).	Automatic selection: prefer SignAndEncrypt > Sign > None
Request timeout in milliseconds	The timeout duration for client requests in milliseconds.	5000
Interval between reconnection attempts (Length)	The length of the interval between reconnection attempts.	3
Interval between reconnection attempts (Unit)	The unit of the interval between reconnection attempts.	Seconds
Maximum number of connection retries	The maximum number of connection retry attempts (0 = unlimited).	0
Client certificate keystore for OPC UA secure channel	The keystore containing client certificates for secure communication.	No client certificate keystore
Display format of OPC-UA NodeIds in Events	**How node ids are handled for event filtering** • Node identifier as human-readable name string • Node identifier in its standard OPC UA parsable format	• Human-readable: `"Temperature Sensor 1"` • Parsable format: `"ns=2;s=MyTemperatureSensor"`
Subscription Groups	Configuration of the subscription intervals (see **Subscription Groups**)	

Subscription Groups

3 predefined subscription groups (default, slow and fast) can be used for subscribing to variable and event nodes.

Property	Description	Example
Publishing interval (Length)	Interval length for publishing data (in milliseconds)	1000 (Default), 10000 (Slow)
Publishing interval (Unit)	Time unit for publishing interval	Milliseconds
Sampling interval (Length)	Interval length for sampling data (in milliseconds)	500 (Default), 10000 (Slow)
Sampling interval (Unit)	Time unit for sampling interval	Milliseconds
Queue size	Size of the monitoring queue	1
Discard oldest	Whether to discard the oldest data when the queue is full	Checked (True)
Monitoring filter	Filter for monitoring specific data	None
Max number of monitored items	Maximum number of items that can be monitored	1000

Events (Alarms)

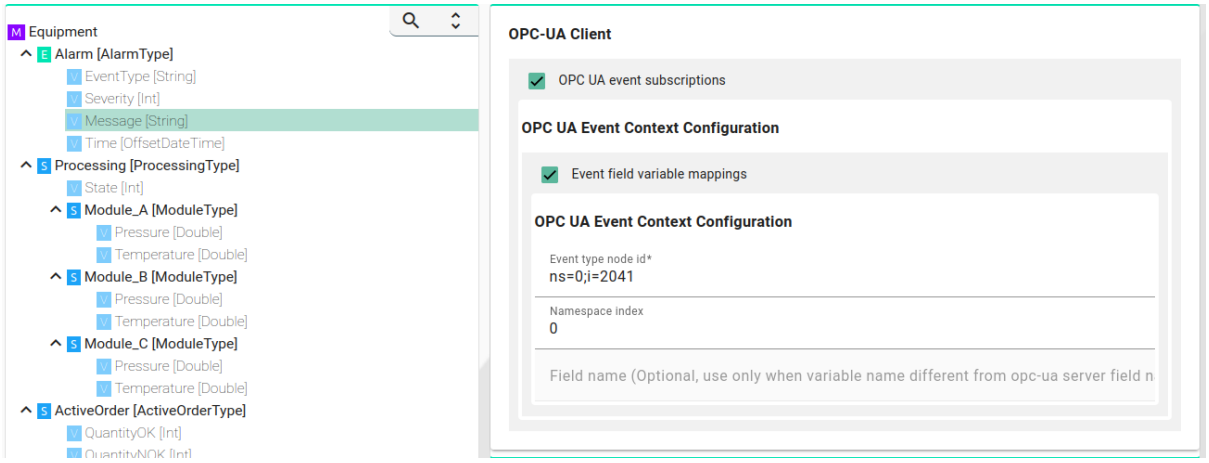


OPC UA Event Context Configuration

Property	Description	Example
Event source node ID	Node ID of the event source in OPC UA	"ns=0;i=2253" (server node ID)
Event subscription group	Group to which the event subscription belongs	"Default"
OPC UA event filter where clause	Filter condition for event subscription	"Severity >= 500"

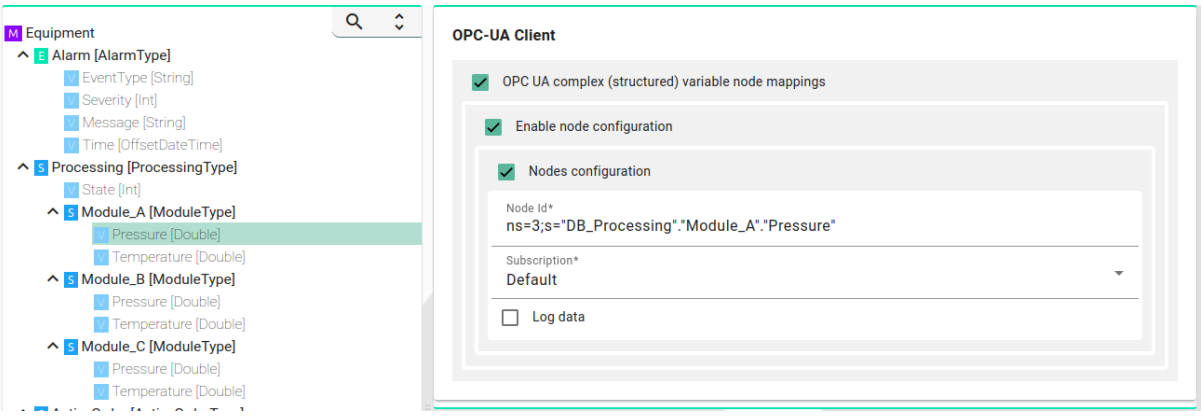
Event variables

Supported event variables are "EventType" (String), "Severity" (String), "Message" (String) and "Time" (OffsetDateTime)



Property	Description	Example
Event source node ID	Node ID of the event source in OPC UA	"ns=0;i=2253"
Namespace index		`0`

Variables



Property	Description	Example
Node Id	Unique identifier for an OPC UA node, including namespace and path.	`ns=3;s="DB_Processing"."Module_A"."Pressure"`
Subscription	Defines the subscription type for monitoring node value changes.	`Default`, `Slow`, `Fast`
Log data	Enables logging of node data for debugging	(Enabled via checkbox)

OPC-UA Server

Characteristics

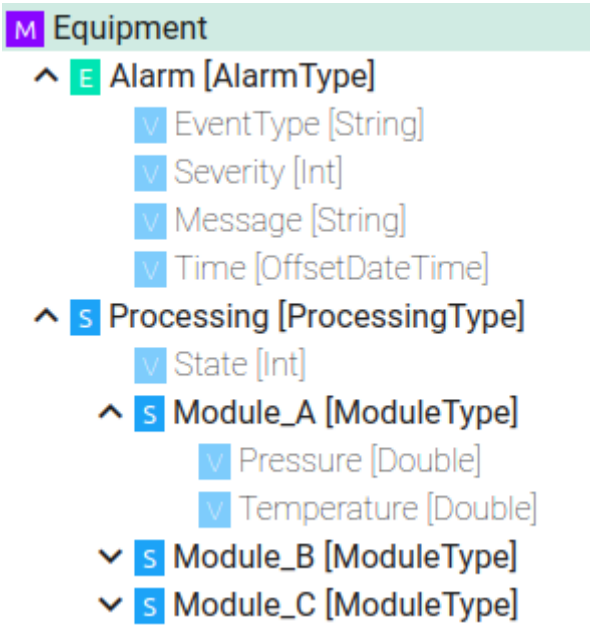
OPC (Open Platform Communications) enables access to machines, devices and other systems in a standardized way. To learn more about the standard visit the OPC-UA website.

Supported features

- Data Access

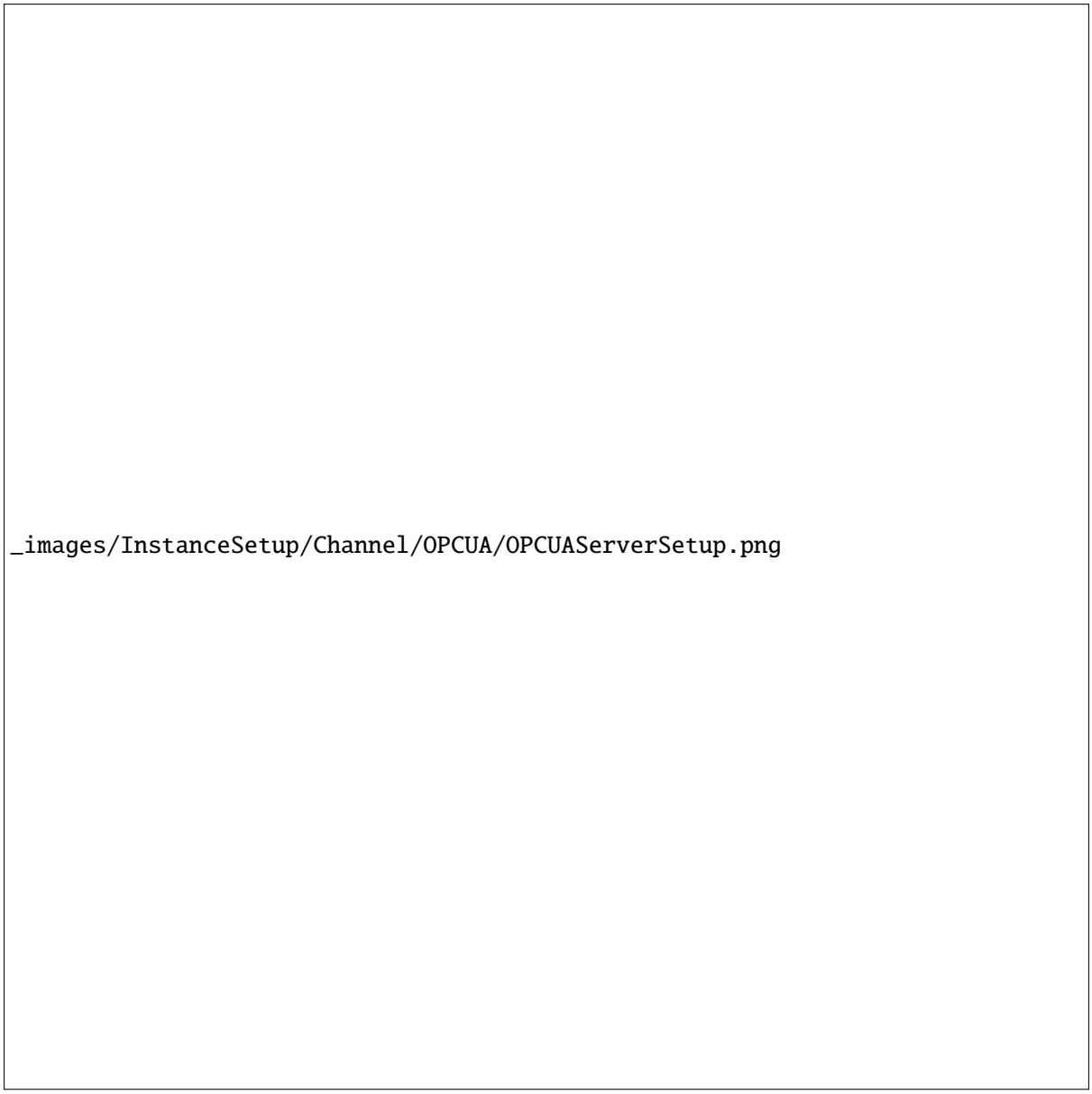
Information Model Requirements

- The following Node Types can be used to model data structures:
  - *Variables* with a *Simple Data Type*.
  - *Variables* with a *Custom Data Type*.

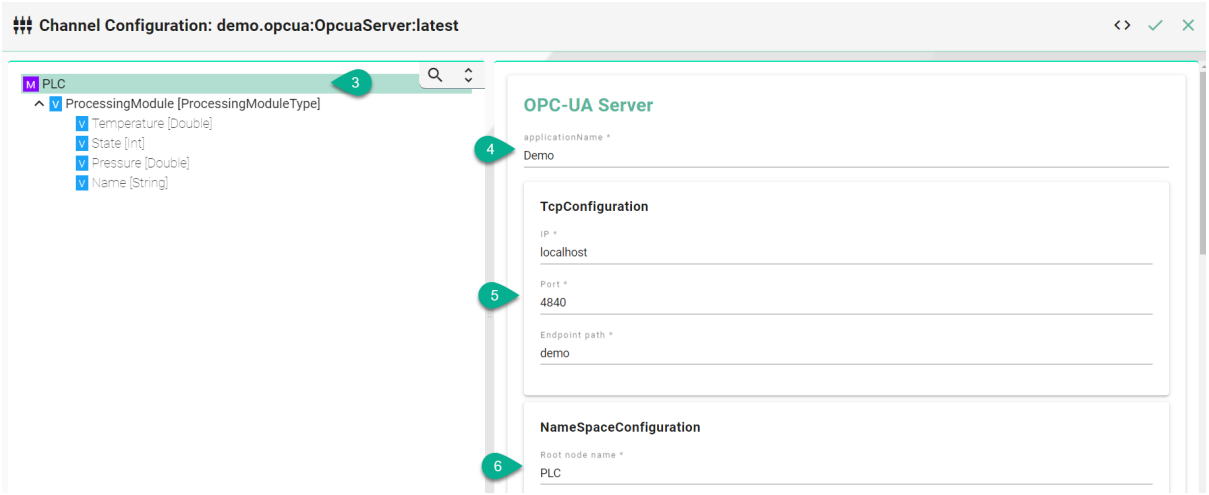


Configuration

1. Select **OPC UA Server** as Channel Type.
2. Click the **Configure** button.



3. Make sure the root model node is selected to configure the OPC-UA Server
4. Enter an **application Name**
5. Configure TCP
   - Enter an **Ip Address**
   - Enter the **Port**
   - Define an **Endpoint**
6. Configure the **NameSpace**
   - Provide a **Root node name**



7. Configure the variable sampling interval
   - Set the **Initial delay** in milliseconds
   - Input the **Sampling rate** in milliseconds

_images/InstanceSetup/Channel/OPCUA/OPCUAServerRootNodeConfig2.png

Description of configuration properties:

Property	Description	Example
IP Address	Server IP	`127.0.0.1`
Port	Server port	`4840`
Endpoint path	Service name at the server endpoint	`demo`
Root node name	The name of the top-level node in the OPC UA server's address space	`PLC`
Initial delay	The time the OPC UA client/server should wait before attempting to connect to the OPC UA server	`10`
Sampling rate	The rate at which the OPC UA client/server requests data from the OPC UA server	`1`

REST

Characteristics - REST

Representational state transfer (REST) is a software architectural style that describes a uniform interface between decoupled components in the Internet in a Client-Server architecture. To learn more about the standard visit the REST section in Wikipedia website.

The characteristics of a REST server that need to be configured typically include the following:

- Host and Port: The network address (IP or hostname) and port number on which the server listens for incomming HTTP requests.
- Base URL/Context Path: The root path under which the REST API is accessible (e.g. /api/v1).
- SSL/TLS (HTTPS): Certificates and protocols used for secure communication.
- Message Encoding: The character encoding used for the body of HTTP requests and responses.
- Timeouts: Limits for request processing time, connection time, etc.
- Endpoint/Routes: The specific URI's and HTTP methods (GET, POST, PUT, DELETE, etc.) that the server will handle.
- Thread Pool/Concurrency Settings: Limits for the number of concurrent requests the server can handle.

Configuration - REST Server

To configure a REST server in SmartUnifier, a few core concepts should be understood:

- Information Model: defines the structure of resources on the server but also detailed configuration of the endpoints/routes (Parametes, Headers, Body structure). In the current version of SmartUnifier only Variables, Complex Variables and Commands can be used for REST server configuration.
- Communication Channel: specifies how the defined structure will be communicated to clients.

Variables and Complex variables can be used together with the rest server as storage for data. For example you have an equipment with all kind of data points like temperature, pressure, etc., these data points can be stored in a complex variable and the rest server can be configured to return the data in a structured way.

Commands can be used to execute actions on the server. Command parameters can be used as headers, url parameters or body content. Command reply can be used retrieve the return code, headers and body of the executed command.

General REST Server channel configuration:

Property	Description	Example
pathPrefix	root path under which the REST API is accessible	`/api/v1`
IP	The IP address of the REST server	`localhost`
Port	The port of the REST server	`8080`
Default content type	The default content type of the REST server. Default value is application/json.	`application/json`
Webapp	When enabled the REST server will be able to be configured to host a web application.	
SSL	When enabled the REST server will be able to be configured to use SSL encryption.	
Message encoding	Encoding standard for messages	`ISO-8859-1`, `UTF-16`
Maximum Handling Time in ms		
Log data	Show message body in logs	
Threadpool size	Number of threads that should handle incomming requests	`0`

Webapp configuration

Property	Description	Example
Warfile	The path to the WAR file of the web application.	c:/helloworld.war
Context path	The path where the web application can be accessed.	/webapp/helloworld

SSL configuration

Property	Description	Example
Keystore	Select to specify where are the SSL certificates stored.	Java Certificate Keystore or Windows Certificate Management
Truststore	Select to specify where are the trusted certificates stored.	Java Certificate Keystore or Windows Certificate Management
Port	Optional configuration to specify https port in case http and https are both enabled	9445

Java Certificate Keystore configuration

Property	Description	Example
Keystore file path	The path to the Java KeyStore file.	`c:/keystore.jks`
Password	The password for the Java KeyStore file.	`password`

Note

To create a test certificate in the Java Certificate Keystore, use the following commands: (change all entries within ' ' to match your specifics)

```
SmartUnifierManager\jre\bin\keytool.exe -genkeypair -keyalg RSA -keysize 2048␣
↪-keystore keystore.jks -alias SmartUnifier3 -dname "CN='smartunifier-https-
↪vhost',OU=amorph.pro,O=Amorphsys,C=DE" -storepass 'changeit' -keypass
↪'changeit' -validity 3650 -ext KeyUsage=digitalSignature,dataEncipherment,
↪keyEncipherment,keyAgreement -ext ExtendedKeyUsage=serverAuth,clientAuth
```

```
SmartUnifierManager\jre\bin\keytool.exe -v -list -keystore keystore.jks -
↪storepass changeit
```

Windows Certificate Management configuration

Property	Description	Example
Certificate Name (optional)	The name of the certificate to use.	
Type	Specifies in which windows store the certificate is located.	Current User MY
Provider	Specifies the cryptographic service provider (CSP) to use for certificate operations.	`SunMSCAPI`

There are 4 supported stores where the certificate can be placed:

- **Local Machine MY** Stores certificates and private keys accessible to all accounts on the system.
- **Local Machine ROOT** Contains root CA certificates and trusted self-signed certificates accessible to all accounts on the system.
- **Current User MY** Stores personal certificates and private keys accessible only to the current user.
- **Current User ROOT** Contains root CA certificates and trusted self-signed certificates accessible only to the current user.

Note

To create a test certificate in the Windows Certificate Store, use the following PowerShell command:

```
New-SelfSignedCertificate -DnsName localhost -CertStoreLocation cert:\
↪LocallMachine\Root
```

> **Note**
>
> Because this certificate is stored in **Local Machine MY**, you need to run the SMART**UNIFIER** as an Administrator to access it.

Variable and Complex Variable configuration

No configurations are needed for the variable and complex variable nodes. Server will automatically create routes based on the pathPrefix and variable path. E.g. if the pathPrefix = /api/v1 and the variable is created in the root of the information model, with the name temperature, the route will be /api/v1/Variable/temperature. If the variable is created under the Measurements complex variable, the route will be /api/v1/Variable/Measurements/Variable/temperature. But also /api/v1/Variable/Measurements route will be available. For both variable types GET, PUT, POST methods will be available.

Command configuration

By default commands are disabled. To enable commands you have to select it in the channel configuration from the information model tree. By checking the "Command routed" you enable that command. If no other configuration is done a default route will be created based on the pathPrefix. E.g. if the pathPrefix = /api/v1 and the command is created in the root of the information model, with the name getTemperature, the route will be /api/v1/Command/getTemperature. As for the variables GET, PUT, POST methods will be available.

Command configuration details

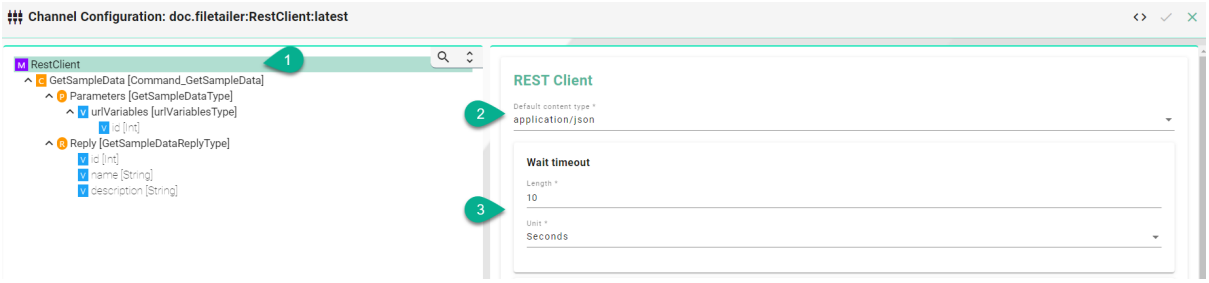
Property	Description	Example
URL	URL of the REST API. URL is relative to the pathPrefix	`http://localhost:8081/api/v1/dataPoint/${id}`
HttpMethod	HTTP method for the action performed by the Client.	`GET`, `POST`, `PUT`
Content Type	Is used to indicate the media type of the resource. Default value is `default` meaning that the content type will be the same as the default content type of the REST server.	`application/json`
reply content type	Is used to indicate the media type of the reply. Default value is `default` meaning that the content type will be the same as the default content type of the REST server.	`application/json`

Configuration - REST Client

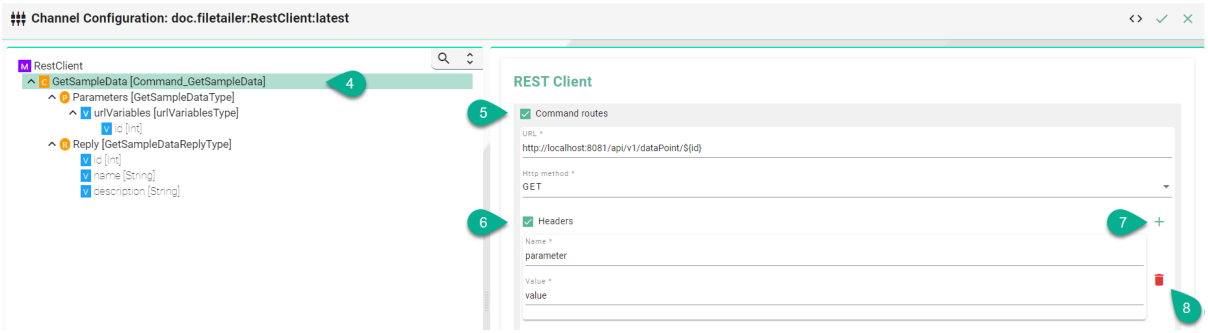
The following sample configuration shows a GET request using url parameters.

1. Select the **root model node** in the tree on the left
2. Select the **content type** - defines the media type of the associated representation

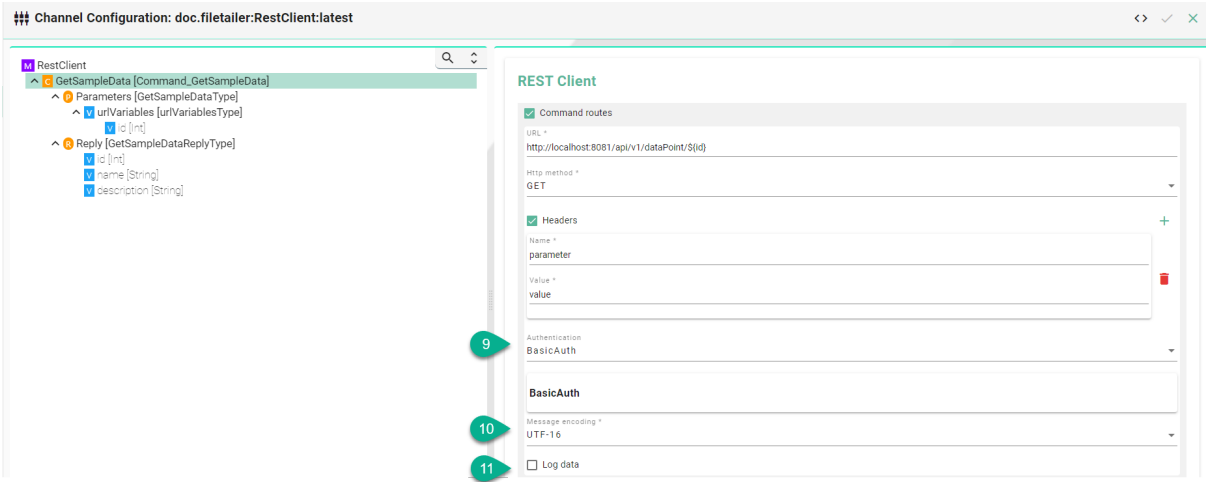
3. Set the **wait timeout**



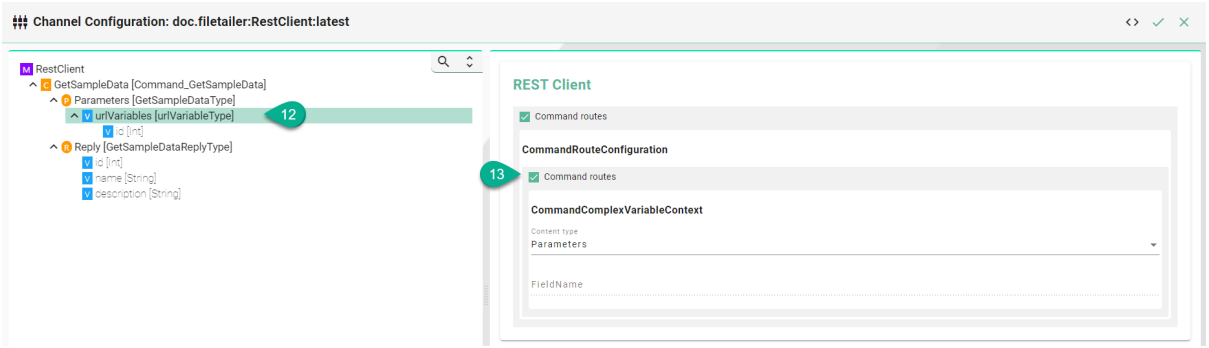
4. Select the **Command** node
5. Enable the **Command routes** checkbox for the configuration of the following fields:
    - Enter the **URL** - If URL parameters are used then add each parameter in the following syntax `${id}`
    - Select the **HTTP method**.
6. Headers (Optional) - Enable the checkbox **Headers** for the configuration of the following fields:
    - Enter the name of the header
    - Enter the value
7. Add multiple header entries by clicking the **Add** button
8. Delete a header by clicking the **Delete** button



9. Select the **Authentication** type
10. Select the **Message encoding** standard
11. Check the box to **Log data** (E.g., the body of a request).



12. URL Parameters (Optional) - Select a custom variable node
13. Enable the **Command routes** for the configuration of the following fields:
    - Select the **Content Type**
    - (Optional) Enter a **Field Name** in case the Information Model Node is not matching the REST API



Description of configuration properties:

Property	Description	Example
URL	URL of the REST API.	`http://localhost:8081/api/v1/dataPoint/${id}`
HttpMethod	HTTP method for the action performed by the Client.	`GET`, `POST`, `PUT`
HeaderName and Header Value	To provide server and client with additional information	`Retry-After: 12`
Default Content Type	Is used to indicate the media type of the resource.	`application/json`
RouteHeaderConfiguration	Headers represent the meta-data associated with the API request	`Name, Value`
Authentication Type	Type of the Authentication	`Basic, Digest, Kerberos, NTLM, SPNEGO`
Content Type of Parameter Nodes	Type of the Parameter	`Parameters, Body, Header, None`
Field Name	For non-matching Information Model nodes and API spelling	`String`
WaitTimeoutDuration	Timeout in seconds until request is failing	`10`
Message encoding	Encoding standard for messages	`ISO-8859-1, UTF-16`

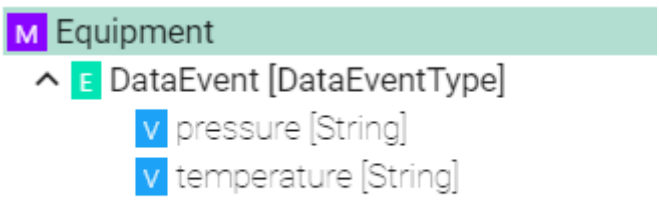
SECS/GEM

Characteristics - SECS/GEM

The SECS/GEM is the semiconductor's equipment interface protocol for equipment-to-host data communications. In an automated fab, the interface can start and stop equipment processing, collect measurement data, change variables and select recipes for products. To learn more about the standard visit the SECS/GEM section in Wikipedia website.

Information Model Requirements

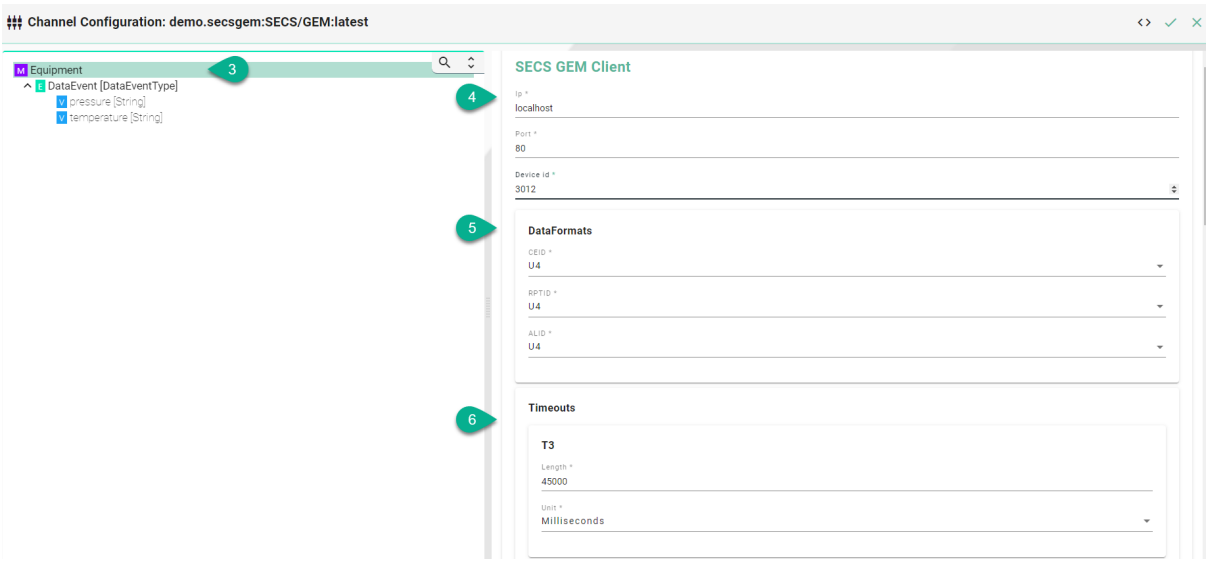
- The first Node after the root node M can be of type *Event* E, *Command* C or *Variable* V
- The following Node Types can be used under the Event Node:
  - *Variables* V with a *Simple Data Type* represents the key-value pairs.
  - *Variables* V with a *Custom Data Type* represent objects that can contain key-value pairs.



Configuration - SECS/GEM Client

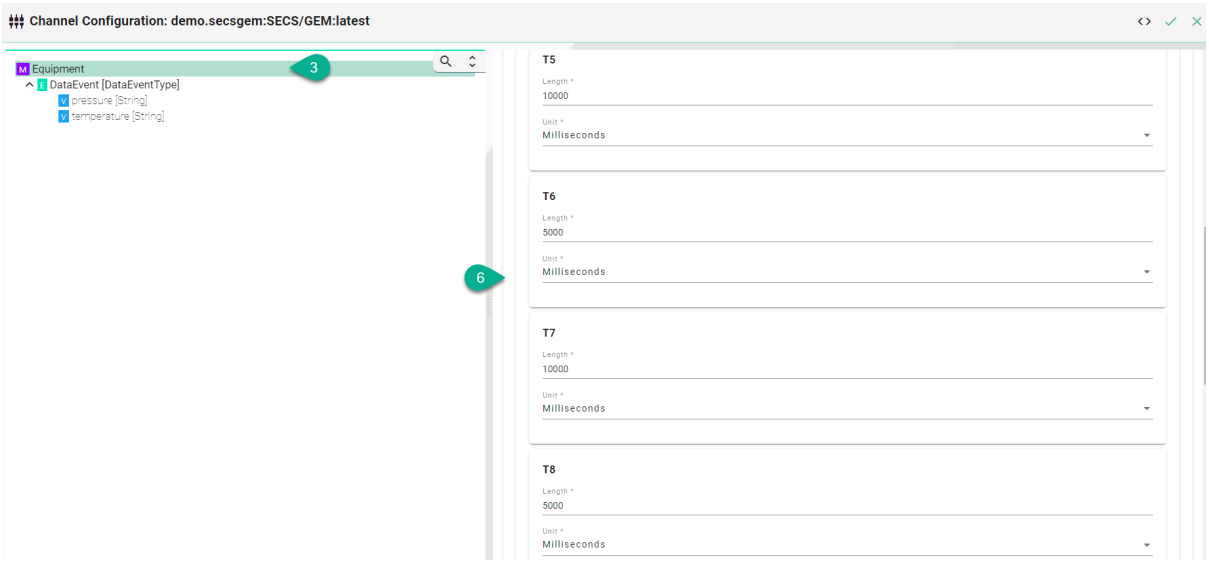
1. Select **Secs Gem Client** as Channel Type.
2. Click the **Configure** button.

3. **Make sure** the root model node is selected to configure the SECS/GEM Client
4. Enter the device configuration:
   - input the equipment-to-host **Ip** address
   - type in the TCP **Port** for the communication
   - input the **Device Id**
5. Enter the **Data Formats**
   - Input **CEID** - format for event Ids
   - Enter **RPTID** - format for report Ids
   - Input **ALID** - format for alarm Ids

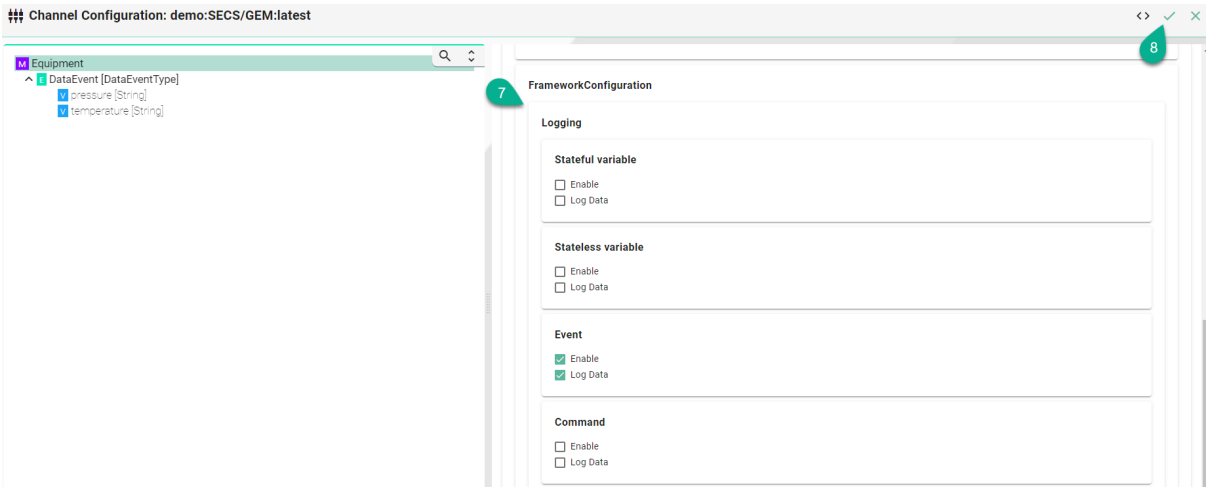


6. Input timeout for:
   - **T3** - Reply Timeout in the HSMS protocol.
   - **T5** - Connect Separation Timeout in the HSMS protocol used to prevent excessive TCP/IP connect activity by providing a minimum time between the breaking, by an entity, of a TCP/IP connection or a failed attempt to establish one, and the attempt, by that same entity, to initiate a new TCP/IP connection.

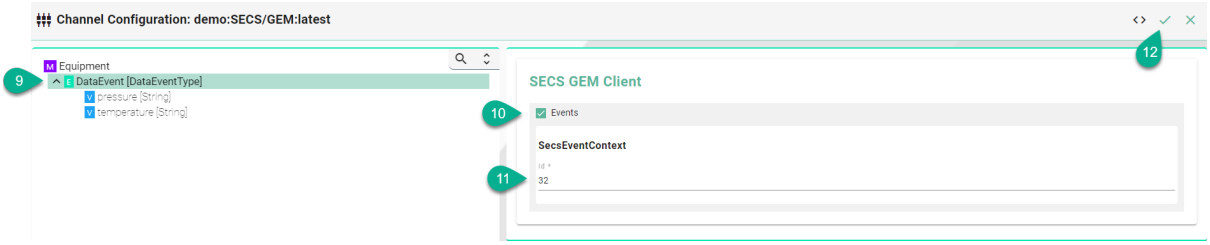
- **T6** - Control Timeout in the HSMS protocol which defines the maximum time an HSMS control transaction can remain open before a communications failure is considered to have occurred. A transaction is considered open from the time the initiator sends the required request message until the response message is received.
- **T7** - Connection Idle Timeout in the HSMS protocol which defines the maximum amount of time which may transpire between the formation of a TCP/IP connection and the use of that connection for HSMS communications before a communications failure is considered to have occurred.
- **T8** - Network Intercharacter Timeout in the HSMS protocol which defines the maximum amount of time which may transpire between the receipt of any two successive bytes of a complete HSMS message before a communications failure is considered to have occurred.



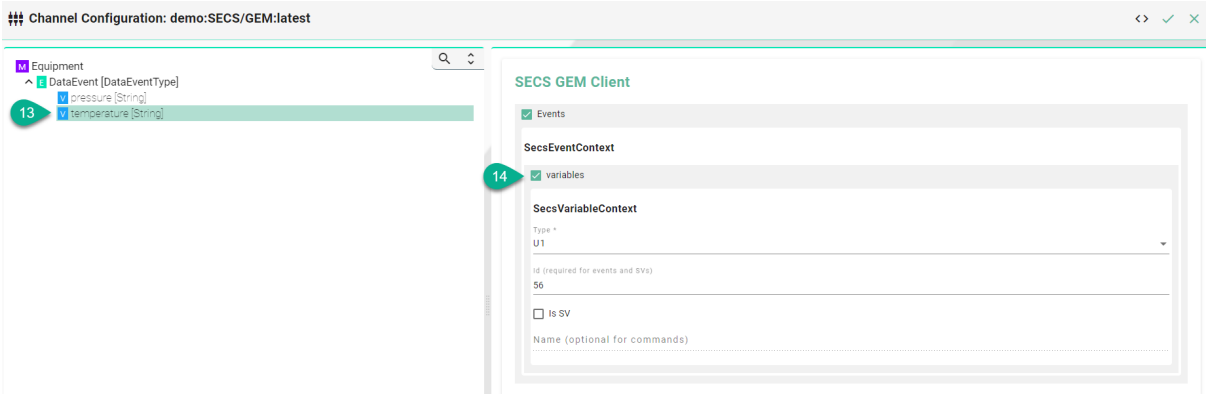
7. Select the logging type for the required Node Types:
   - Check the **Enable** box
   - Check the **Log Data** box



8. Click on the **Apply** button
9. Select the **Event** node to configure the event context



10. Click to check the **Events** box
11. Enter the event context **Id** which will trigger the event in the Information Model
12. Click on the **Apply** button
13. Select the variable in the tree



14. Click to check the **variables** box and configure the Secs variable context
    - select the variable **Type**
    - enter the variable **Id**
    - click the **Is SV** box to check if the variable is a SV
    - input the variable **Name**

Description of configuration properties:

Property	Description	Example
Ip	IP address of the Equipment	`http://localhost`
Port	TCP port for the communication	`5000`
Device Id	Id of the equipment	`NJ-300`
CEID	Format for event Ids	`U4`
RPTID	Format for report Ids	`U4`
ALID	Format for alarm Ids	`U4`
Timeouts	Time interval the connection times out in milliseconds	`45000`
T3	Reply timeout in the HSMS protocol	`10000`
T5	Connect Separation Timeout in the HSMS protocol	`5000`
T6	Control Timeout in the HSMS protocol	`10000`
T7	Connection Idle Timeout in the HSMS protocol	`5000`
T8	Network Intercharacter Timeout in the HSMS protocol	`10000`
Id	Id of the equipment event which will trigger the event	`E32`
Type	Type of variable	`U1`
Id	Variable Id	`V56`
Type	Commands - Type of the message	`S2F41`
Id	Commands Id	`C33`
RCMD	Name of command if it is different from the command Id	`C1`

Email

Characteristics - Email

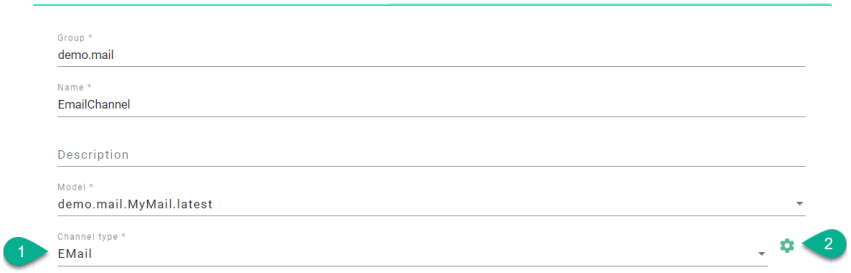
SMART**UNIFIER** provides the capability of integrating the email protocols. The email protocols define the mechanism of the email exchange between servers and clients. An email protocol is a group of rules which ensure that emails are properly transmitted over the Internet.

Information Model Requirements

- The following Node Types can be used to model a register:
  - *Variables* with a *Simple Data Type*.
  - *Variables* with a *Custom Data Type*.

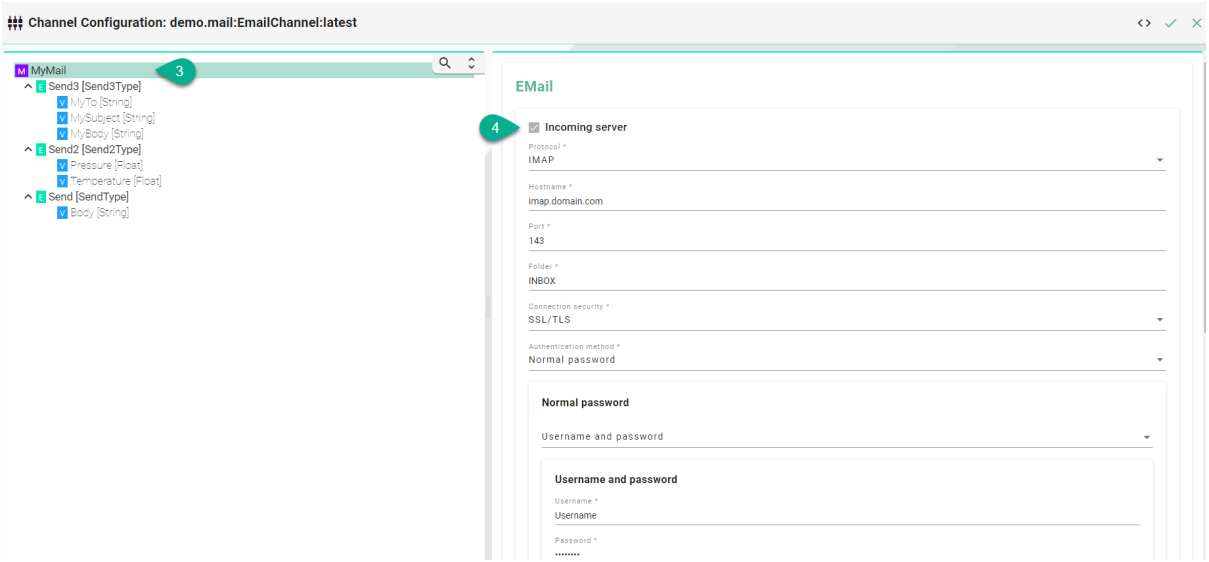
Configuration - Email

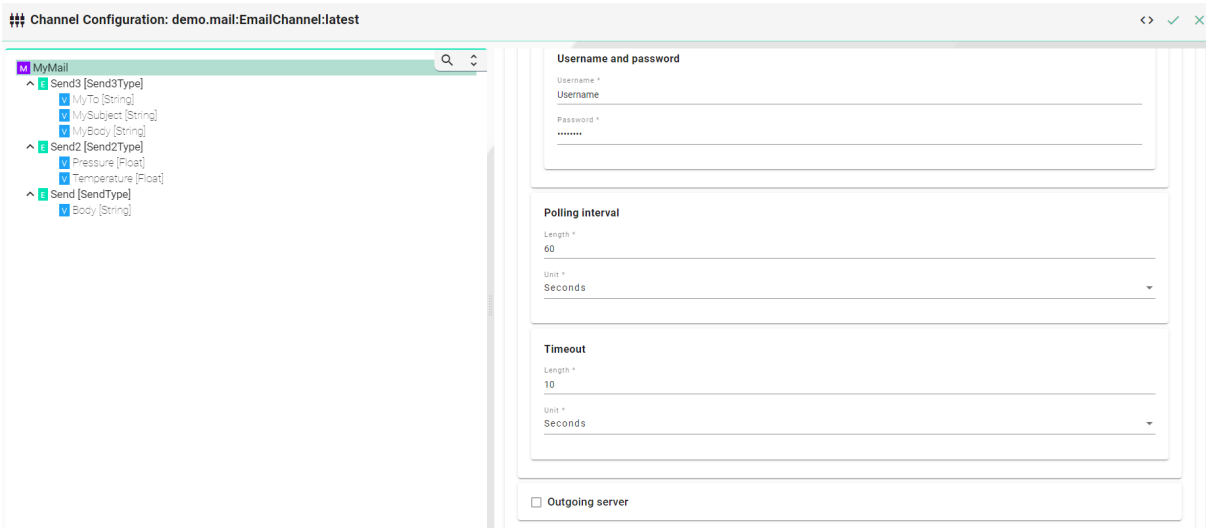
1. Select **EMail** as Channel Type.
2. Click the **Configure** button.



3. Make sure the root model node is selected to configure the Email Client.

4. Enable **Incoming server** for configuration, based on the email provider:
   - Select the **Protocol**
   - Input the **Hostname**
   - Provide the **Port**
   - Input the **Folder** name
   - Select the **Connection security**
   - Choose the **Authentication method**
   - Input credentials
   - Configure the **Polling interval** for checking new emails
   - Configure the **Timeout** length

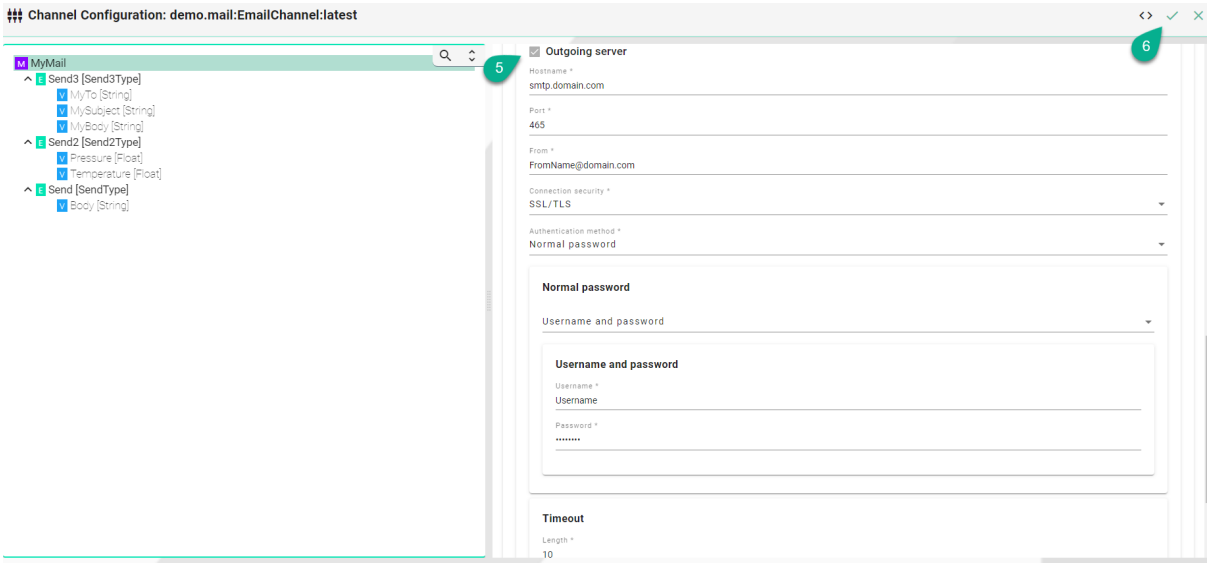




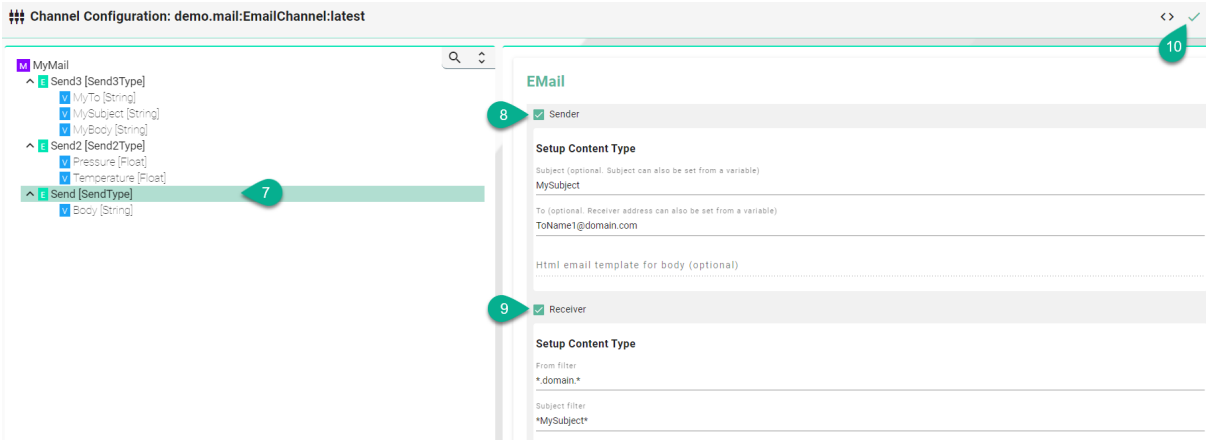
5. Enable **Outgoing server** for configuration, based on the email provider:
   - Input the **Hostname**
   - Provide the **Port**

- Input the **From** hostname
- Select the **Connection security**
- Choose the **Authentication method**
- Input credentials
- Configure the **Polling interval** for checking new emails
- Configure the **Timeout** length

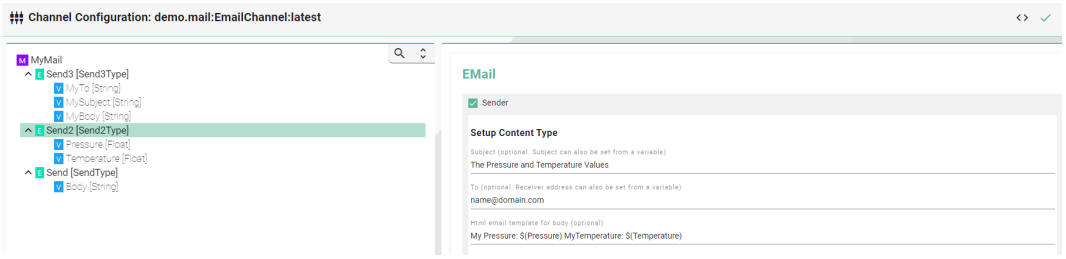
6. Click on the **Apply** button.



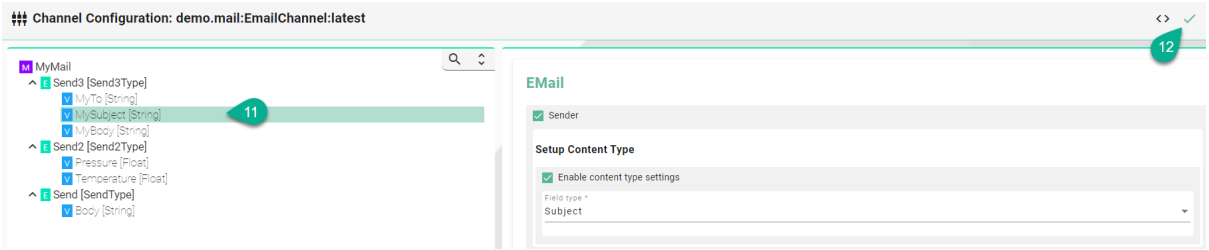
7. Select the Event node.
8. Enable **Sender** for configuration:
   - Input the **Subject** if not using a **Subject** variable under the Event node
   - Provide the **Receiver address** if not using a **To** variable under the Event node
   - Input the **Html email template** (optional)
9. Enable **Receiver** for configuration:
   - Input filter based on the Sender
   - Provide filter based on the Subject
10. Click on the **Apply** button.



- Example of sending the value of a Variable:



11. Select the Variable:
    - If the variable is using a key name (To, From, Subject, Body) no additional configuration is needed
    - Example of Variable used as **Subject**:



12. Click on the **Apply** button to finish.

Description of data type format:

Data Type	Size	Range
BYTE, USINT, UInt8	8 Bit	0 - 255
WORD, UINT, UInt16	16 Bit	0 - 65.535
DWORD,UDINT, UInt32	32 Bit	0 - 4.294.967.295
LWORD,ULINT, UInt64	64 Bit	0 - 2^64-1
SINT, Int8	8 Bit	-128 - 127
INT, Int16	16 Bit	-32.768 - 32.767
DINT, Int32	32 Bit	-2.147.483.648 - 2.147.483.647
LINT, Int64	64 Bit	-2^63 - 2^63-1
REAL, Float32	32 Bit	-3,402823e+38 - 3,402823e+38
LREAL, Float64	64 Bit	-1,7976931348623158e+308 - 1, 7976931348623158e+308

Description of configuration properties:

Property	Description	Example
Protocol	Incoming server protocol	IMAP
Incoming Hostname	Incoming server address	imap.domain.com
Port	Server port	143
Folder	Incoming emails folder	INBOX
Connection security	Communication security standards	SSL/TLS
Polling length	Automatic polling of Email server	60
Timeout length	Time interval the connection times out	10
Outgoing Hostname	Outgoing server address	smtp.domain.com
Outgoing From	Sender Email address	name@domain.com
Subject	Email subject	MySubject
To	Receiver address	name@domain.com
Html email template for body	HTML code for email body	MyTemperature: $(temperature)
From filter	Filter by sender using regex	*.domain.*
Subject filter	Filter by subject using regex	*MySubject*

File Formats / Layers

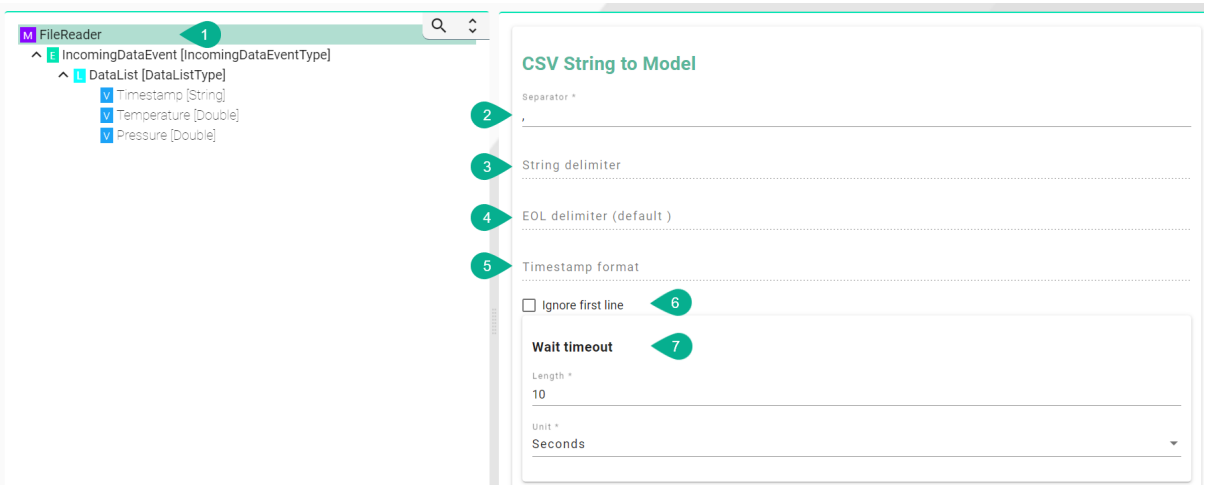
CSV

CSV to Model

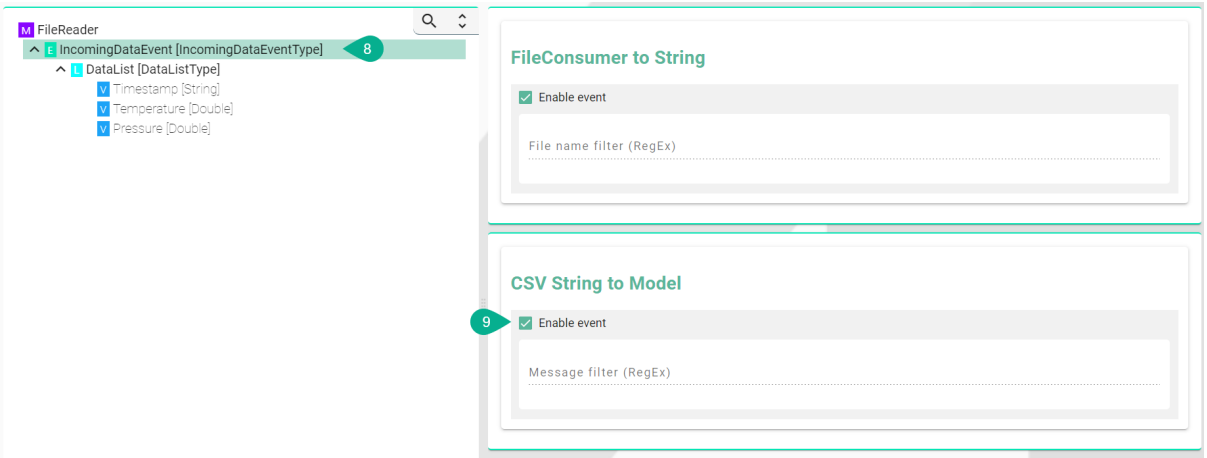
The CSV to Model layer converts CSV data into a structured model representation. It allows to configure various options such as separator, string delimiter, and timestamp format for parsing the CSV file.

1. Ensure the **root model node** is selected in the Information Model.
2. Enter the **Separator** used in the CSV-file.

3. (Optional) Enter a **String delimiter** used in the CSV-file.
4. (Optional) Enter a **EOL delimeter**.
5. (Optional) Enter a **Timestamp format**.
6. (Optional) Check the **Ignore first line** checkbox if the first line of the CSV-file contains the column names.
7. If necessary, adjust the default **Wait timeout** interval.



8. Select the **Event node** in the Information Model.
9. Check the **Enable Event** checkbox if you want to set a filter using a regular expression.



JSON

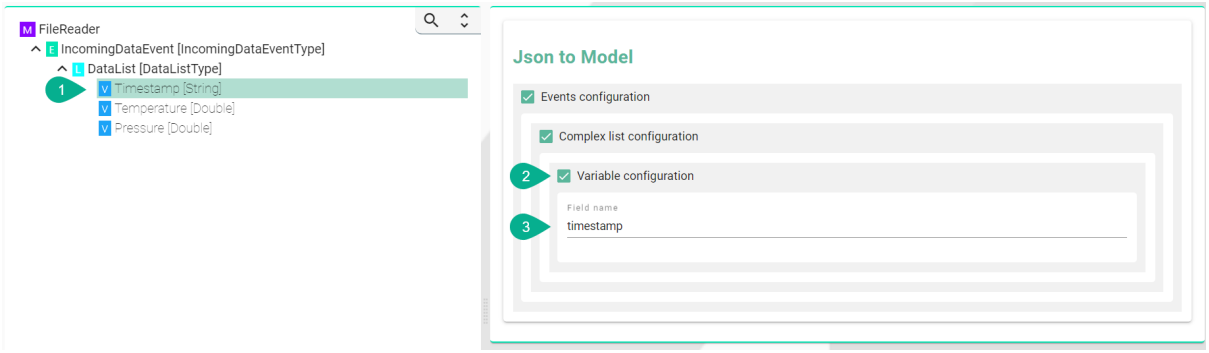
JSON to Model

The naming of JSON elements can be changed if necessary, for example, if the name is not permissible as a variable node name. This applies to both incoming JSON data (e.g., when using the File Reader) and outgoing JSON data (e.g., when using the File Writer).

To change the name of a key in the JSON, the following steps are necessary:

1. Ensure the variable is selected in the Information Model (Parent Nodes have to be enabled as well).
2. Select the checkbox **Variable Configuration**

3. Enter a new name in the **Label** field



XML

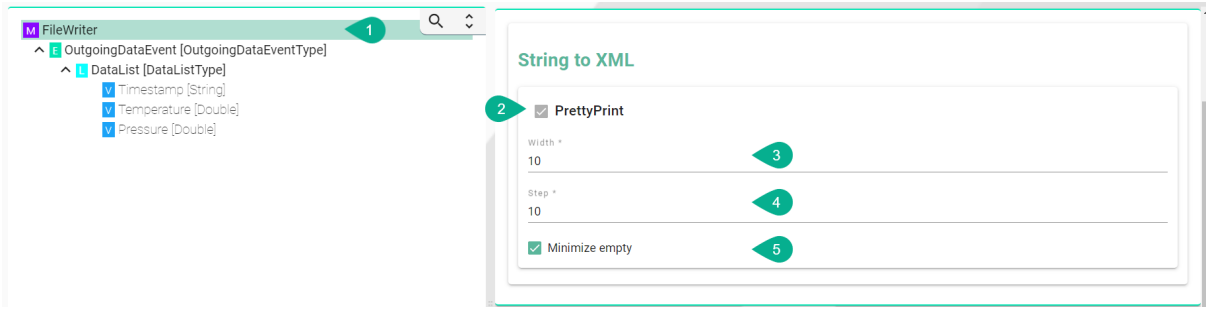
String to XML

Pretty Print

When outputting XML data (e.g., when using the File Writer), it is possible to pretty print the XML data.

To enable pretty printing, the following steps are necessary:

1. Ensure the root model node is selected in the Information Model.
2. Select the **Pretty Print** checkbox.
3. Enter a value for **Width**. The width parameter defines the desired maximum width of the output string in characters.
4. Enter a value for **Step**. The step parameter controls the indentation step, specifying the number of spaces used for each level of indentation in the output.
5. (Optional) Select **Minimize Empty**. When selected, the pretty printer will use the full opening and closing tag format (<element></element>) for empty elements.



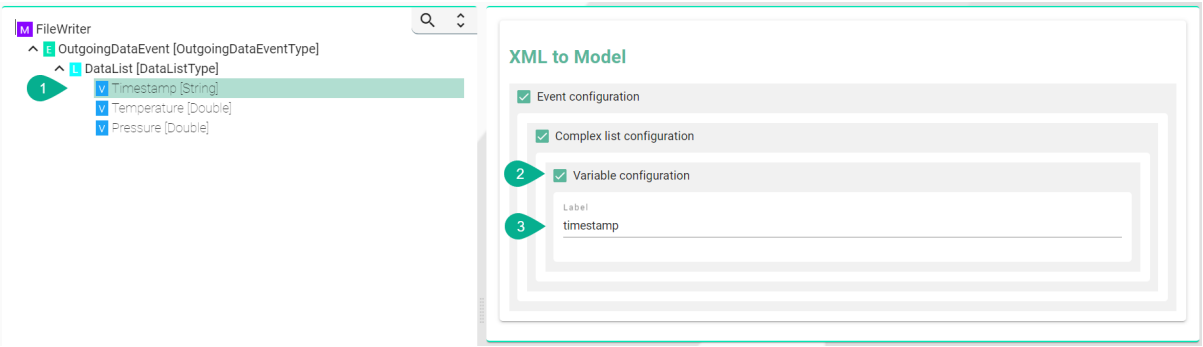
XML to Model

The naming of XML elements can be changed if necessary, for example, if the name is not permissible as a variable node name. This applies to both incoming XML data (e.g., when using the File Reader) and outgoing XML data (e.g., when using the File Writer).

To change the name of an XML element, the following steps are necessary:

1. Ensure the variable is selected in the Information Model (Parent Nodes have to be enabled as well).

2. Select the checkbox **Variable Configuration**
3. Enter a new name in the **Label** field



General Configurations

These configurations apply for all Communication Channel Types.

Framework Configuration

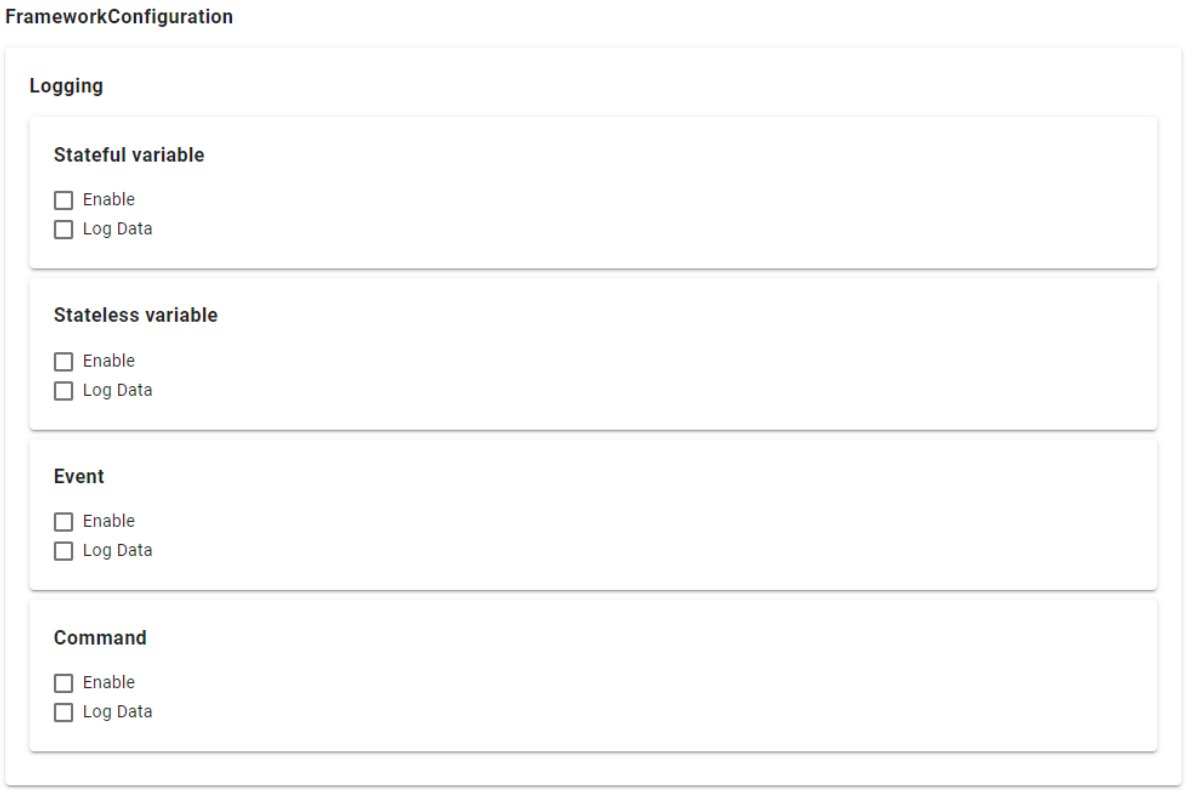
The Framework Configuration enables insights into data handled by *Mapping Rules*. If enabled, logs will be generated once Rules are triggered and executed. These logs are visible then by default in the **INFO** *Log Level* as well as in the *Log Viewer*.

The following Framework Logging Configurations are available:

- Stateful Variable
- Stateless Variable
- *Event*
- Command

For each configuration there are two ways to use logging:

- **Enable**: Logs out information about the *Node Type* that was executed by the Rule.
- **Log Data**: Logs out in JSON-format the actual data of the *Node Type* that was executed by a Rule.



Event Logging

To use the Event Logging enable the checkbox **EventLogging** and for more detailed logging **EventDataLogging**.

Event Logging Output

```
[INFO ] - EventDefinition - Received Event: /Model/bcdbbfd3-cdbe-4ade-8a73-
↪3788e6815c46/Event/ReleaseOrder
```

Event Data Logging Output

```
[INFO ] - EventDefinition - Received Event: /Model/bcdbbfd3-cdbe-4ade-8a73-
↪3788e6815c46/Event/ReleaseOrder={"Quantity":10,"ProductNumber":"Mv5",
↪"OrderNumber":"Ord154","EquipmentId":"4-SWC2"}
```

Mappings

What are Mappings

Mappings represent the SMART**UNIFIER** component that defines when and how data exchange or transformation occurs between two or more *Information Models*. Essentially, it acts as a translator between different Information Models.

A Mapping consists of one or more Rules. A Rule is made up of a Trigger and a list of Actions:

Trigger: specifies when the data exchange or transformation occurs. It can be an element in the Information Model or time-based (e.g., every 5 minutes). See the full list of Triggers below:

- Node Types in the Information Model such as:
  - Variables
  - Events
  - Commands
  - Properties
  - Arrays
- Schedulers, which are time-based triggers, offer the following options:
  - Fixed Rate - The rule is triggered at a fixed interval.
  - Fixed Delay - The rule is triggered at a fixed interval with an initial delay.
  - Timeout - The rule is triggered once after a specified delay.

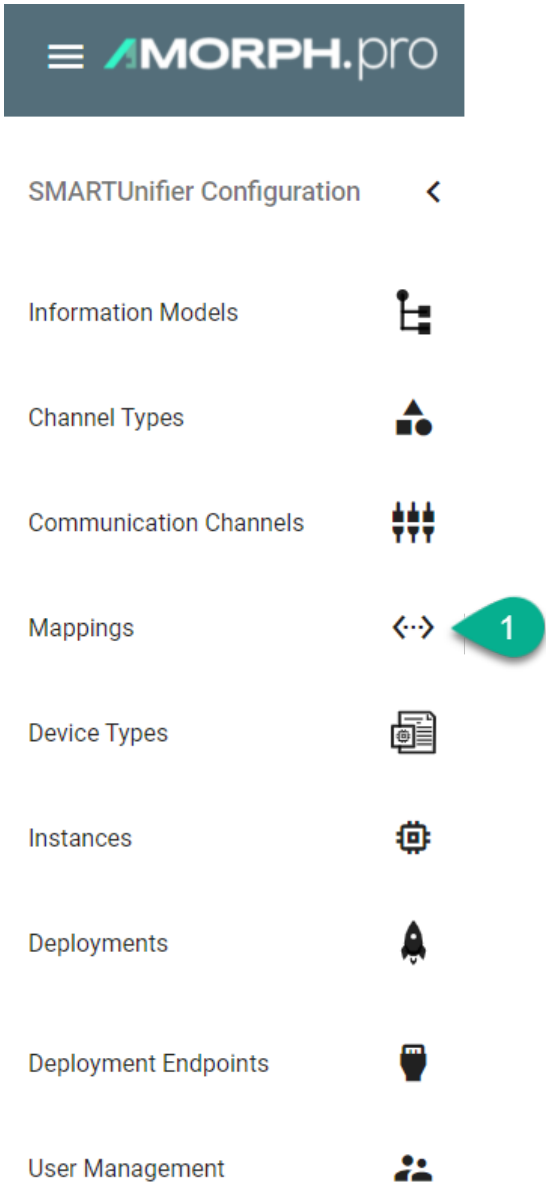
How the different triggers are used is explained in the following sections on *Graphical* and *Code-based* approaches to create a Rule.

Actions: define how the data is mapped from source Node Types to target Node Types.

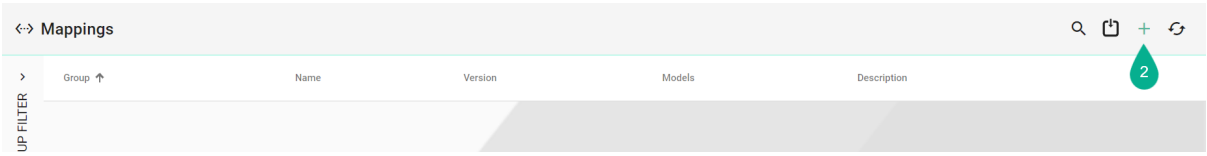
How to create a new Mapping

Follow the steps below to create a new Mapping definition:

- Go the Mappings perspective by clicking the "Mappings" button **(1)**



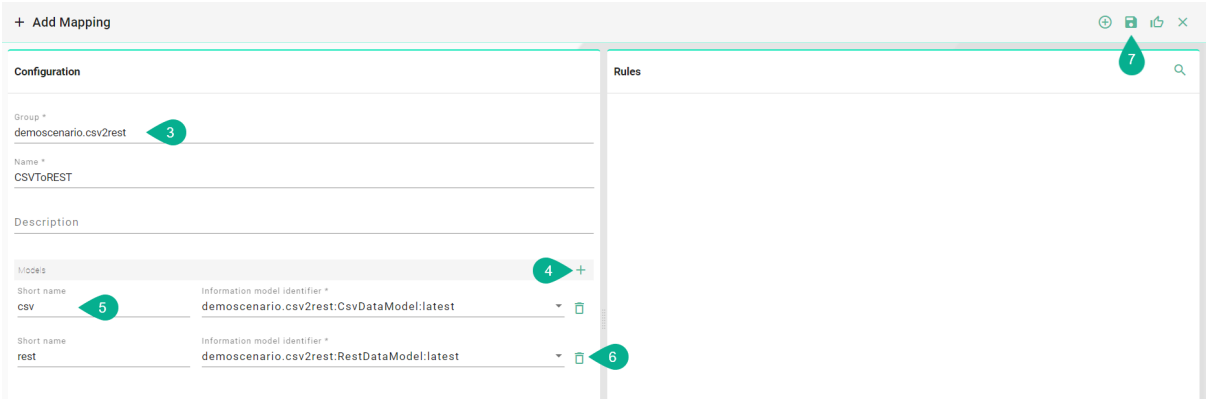
- The following screen displays a list view of existing mappings.
- To add a new mapping, select the "Add Mapping" button located in the top right corner **(2)**.



- On the subsequent screen, provide the following mandatory information: Group, Name, and Version. A description, which is optional, can also be added **(3)**.
- Click the "Add Model" button **(4)**.
- Select the Information Model for this mapping and enter a short name for it. The short name is used to access elements within the model. **(5)**
- To remove a model from the mapping, click the "Remove Model" button **(6)**. This action is

only possible if elements of the model are not used within the mapping's rules. To delete a model, ensure that you first remove its elements from the rules.

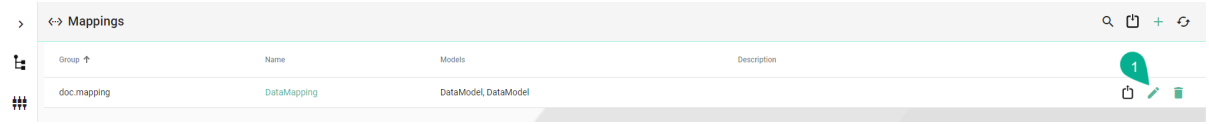
- Once all mandatory fields have been filled in, the "Save" button located in the top right corner will be enabled. Click this button to submit the new *Mapping* **(7)**.
- The newly created mapping is now visible in the list view.



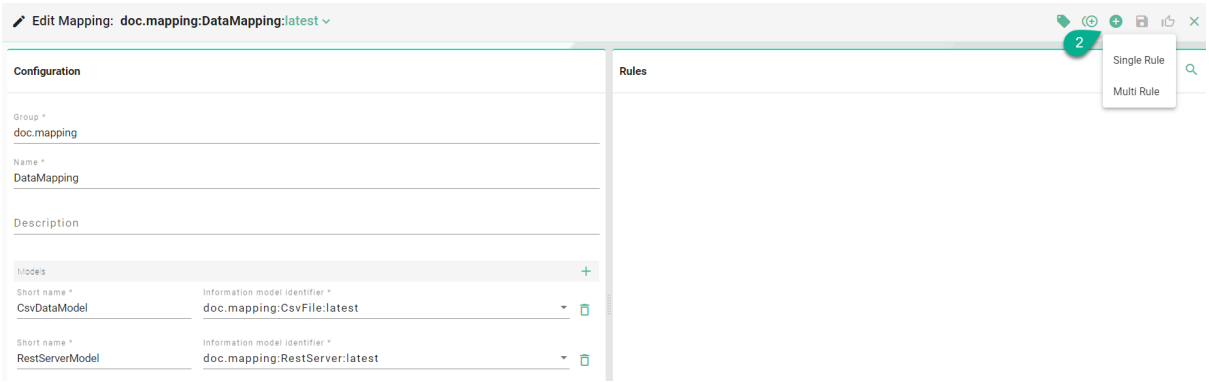
How to create Rules

To create *Rules*, follow the steps described below:

- Select the "Edit" button **(1)**.



- Select the "Add Rule" button located in the top right corner **(2)**.
- Two options are available:
  - **Single Rule** - Defining a single trigger and a list of actions. The Trigger can be an element in the Information Model or time-based (e.g., every 5 minutes).
  - **Multi Rule** - Defining multiple triggers, each with assignments between source and target elements. The Trigger is the change of a source element in the Information Model.



Rule Naming Convention

The name of the rule should reflect the logic that is executed when the rule is triggered. Below are some examples:

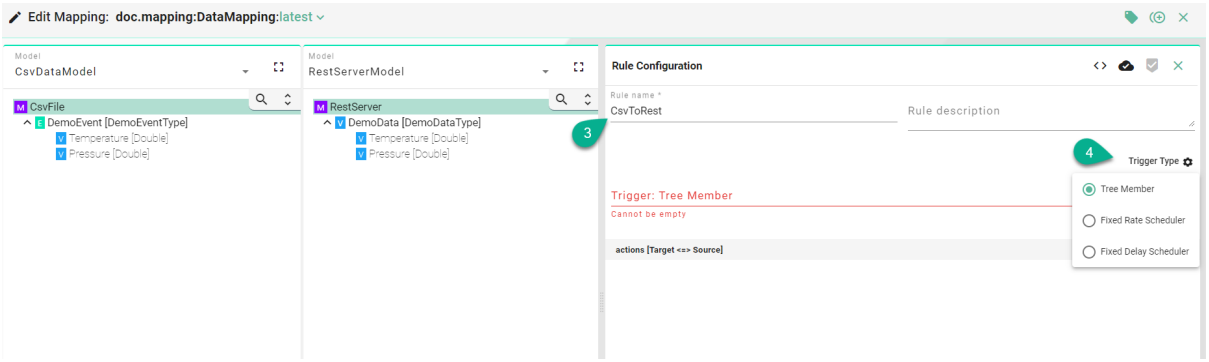
Example Scenarios	Good Naming
Inserting data into a database	Database_Insert
Executing a POST/PUT request on a REST server	Update_Data
Reacting to an input (e.g., StartOrder button on a MES)	StartOrder

Graphical

When creating rules using the Graphical Editor, a rule type needs to be selected first: **Single** or **Multi Rule**.

Single Rule

- The following screenshot shows the Single Rule Editor. The rule contains the following components:
  - Name and an optional Description field.
  - Trigger which can be either a Tree Member of the Information Model, Fixed Rate Scheduler or Fixed Delay Scheduler.
  - Actions where source-to-target assignments are defined betweent the elements of the Information Models.
- Enter "Rule name" **(3)**.

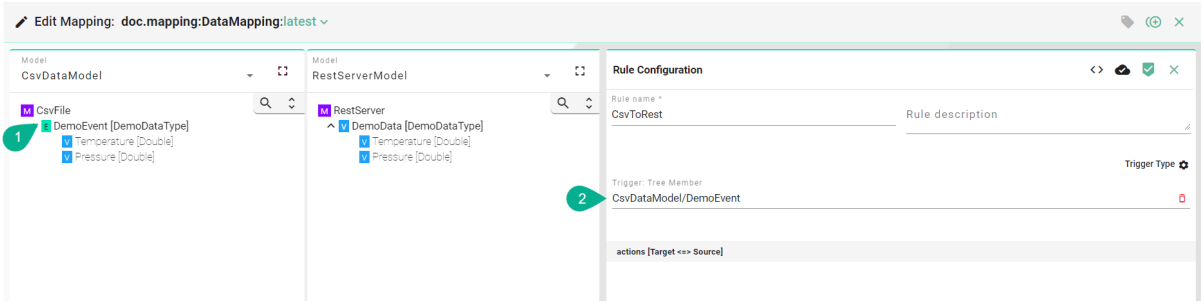


- Select the "Trigger Type" **(4)**:
  - Tree Member - rule with an Information Model tree member as trigger.
  - Fixed Rate Scheduler - rule with a time based trigger, using a Cron Expression.
  - Fixed Delay Scheduler - rule based on a scheduled delay.
  - Timeout Scheduler - rule based on a timeout.

Trigger Types

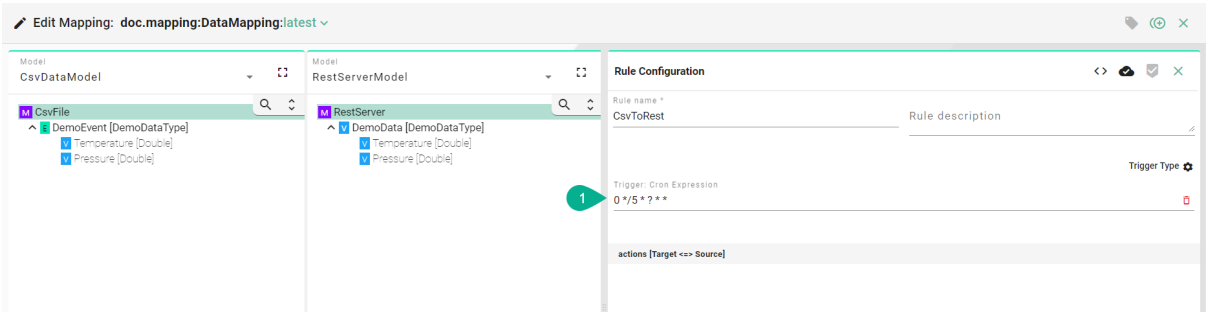
Tree Member

- Drag and drop the *Trigger* from the model panes **(1)** into the trigger field **(2)**.



Fixed Rate Scheduler

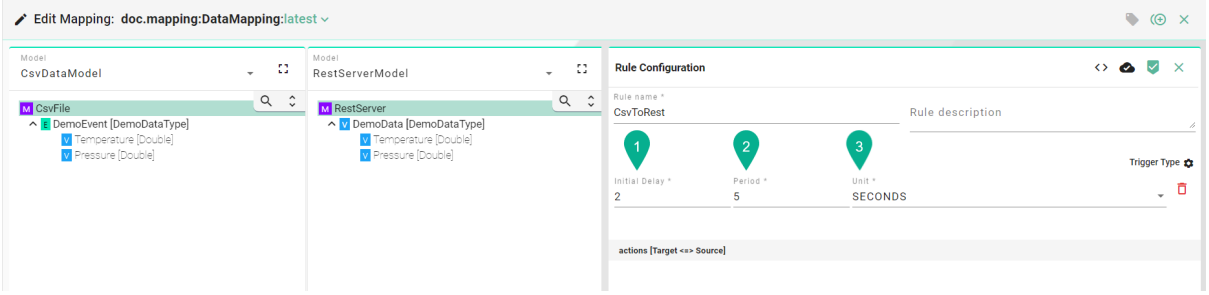
Input a "Cron Expression" **(1)** to set the time based trigger. (E.g., `0 */5 * ? * *` meaning the trigger is set at every 5 minutes).



> **Hint**
>
> Cron expressions are primarily designed for specifying schedules in terms of seconds, minutes, hours, days, months, and weekdays. They do not support specifying intervals at the millisecond level. To use millisecond intervals, use the Fixed Delay Scheduler.

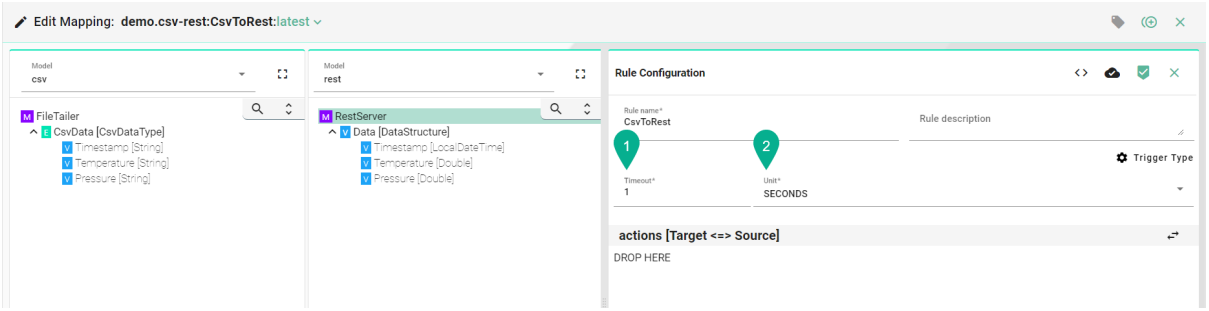
Fixed Delay Scheduler

Input the trigger "Initial start Delay" **(1)**, the "Period" delay **(2)** and the "Unit" **(3)**.



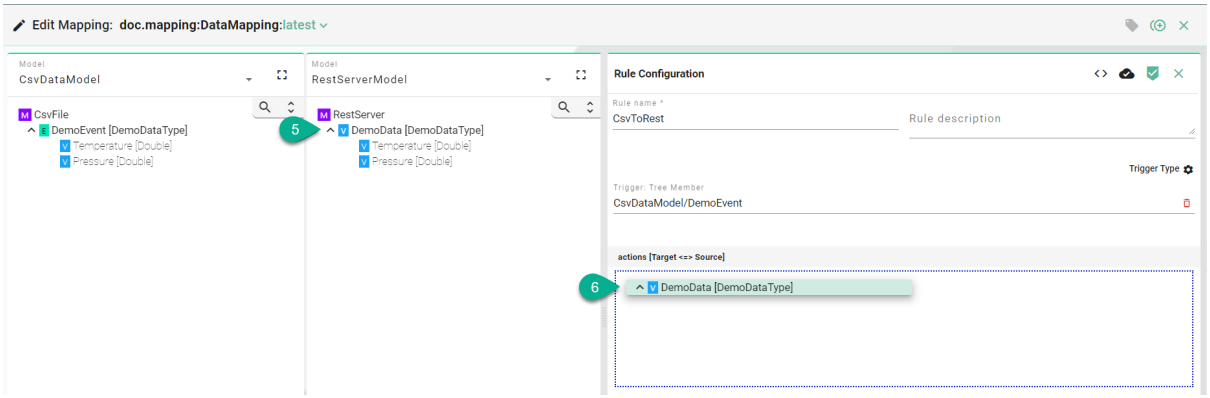
Timeout Scheduler

Input the trigger "Timeout" **(1)** and the "Unit" **(2)**.



Actions

Drag and drop the *Target* Information Model node **(5)** into the Target field **(6)**.

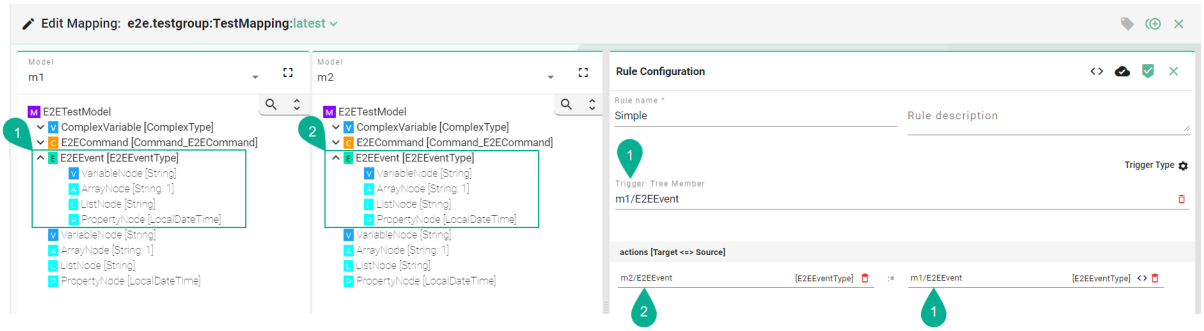


A popup appears to select the assignment type:

- Simple - assignment of a source element to a target element both having the same custom data type - no need to assign each children elements separately
- Complex - assignment of a source element to a target element both having different custom data types

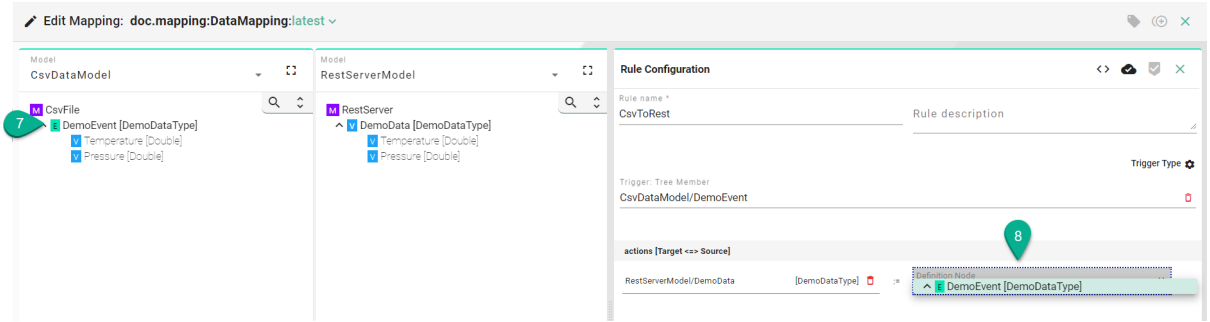
Simple Assignment

When Source and Target are of the same *data type* they can be directly assigned to one another.



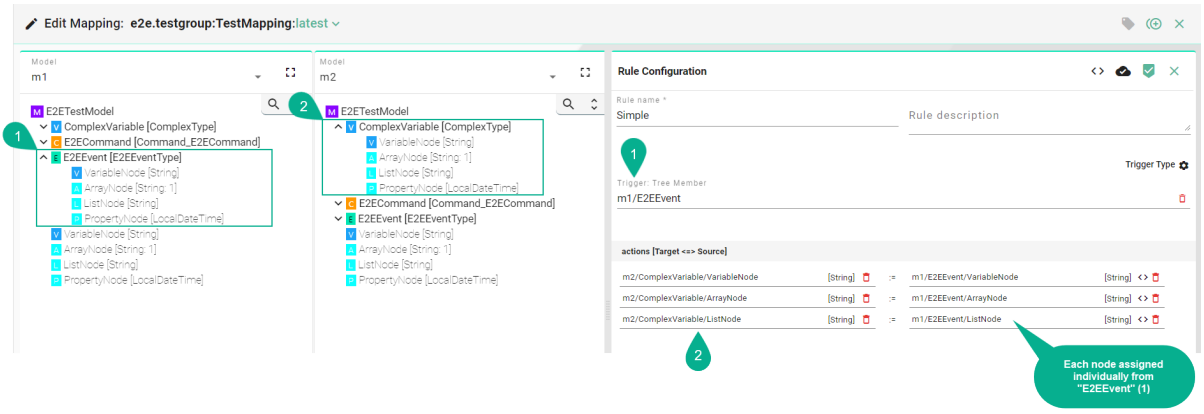
Follow the steps below to create a simple assignment:

Drag and drop the *Source* Information Model node **(7)** into the Source field **(8)**. The Source and the Target node data type must be matched one on one (e.g., DemoEventType to DemoEvent-Type).



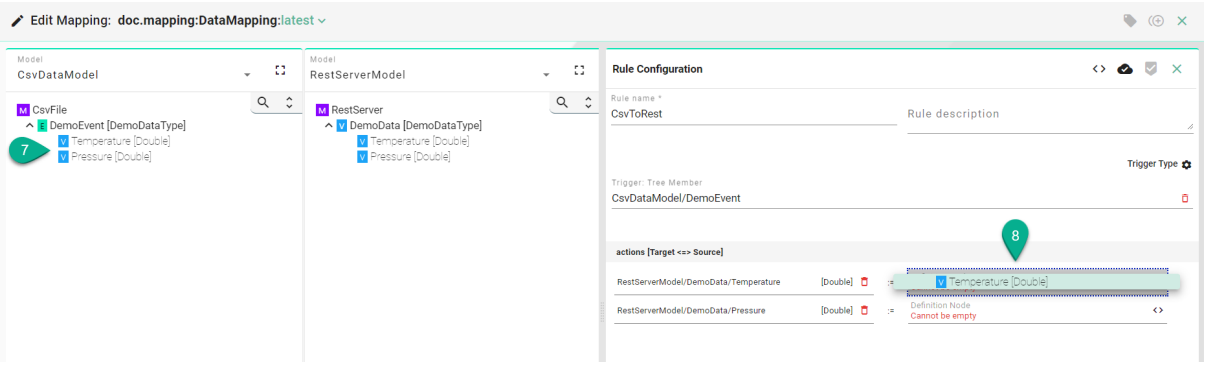
Complex Assignment

When the Source and Target differ in *data type*, their child *nodes* must be assigned individually.



Follow the steps below to create a complex assignment:

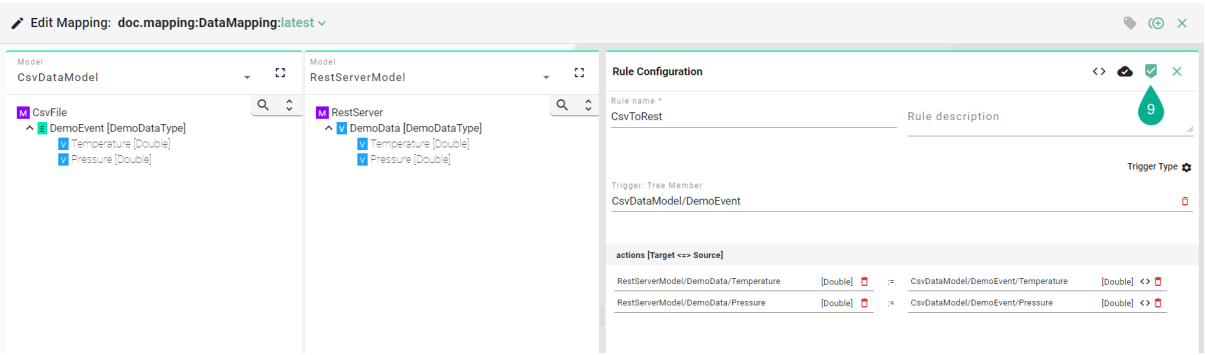
Drag and drop the *Source* Information Model node children's **(7)** one by one into the Source field **(8)**.



Hint

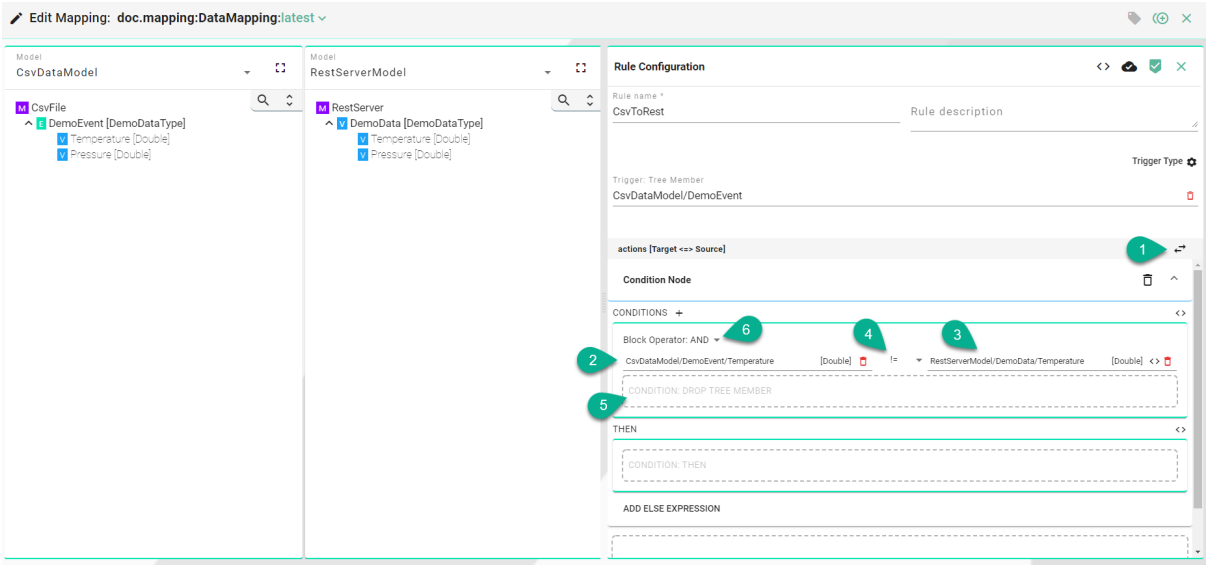
If the Source and Target have different data types (e.g., Int and String), a type conversion can be performed by using *conversion functions* .

- After all mandatory fields have been filled out, select the "Apply" button **(9)** to save the newly created Rule.
- The Single Rule Editor is closed and the newly created Rule is displayed in the Rules List.
- Select the "Save" button placed in the upper right corner to save the Mapping.

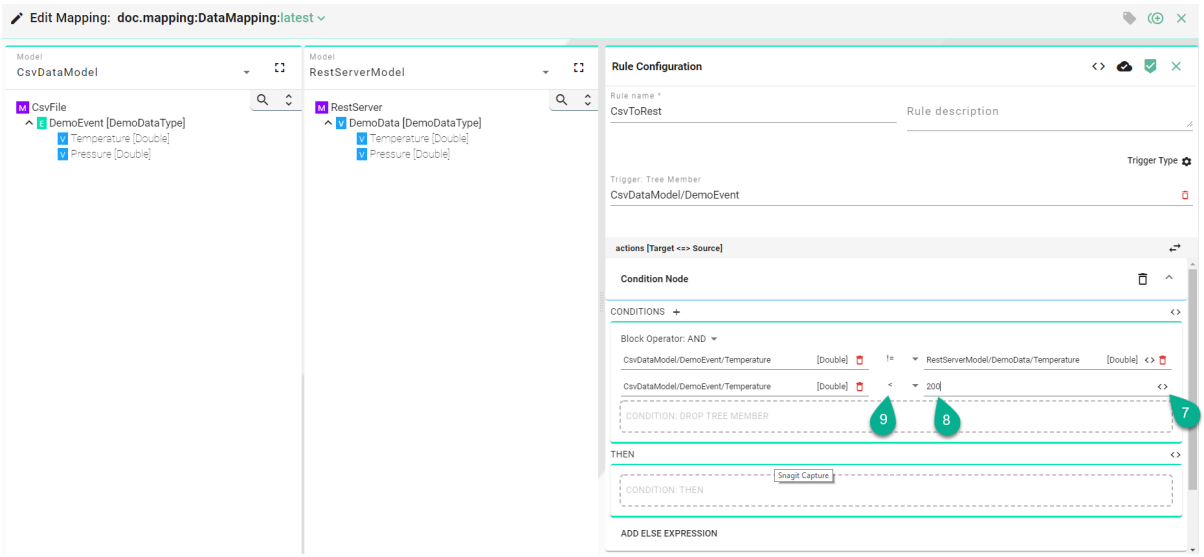


Actions with Conditions

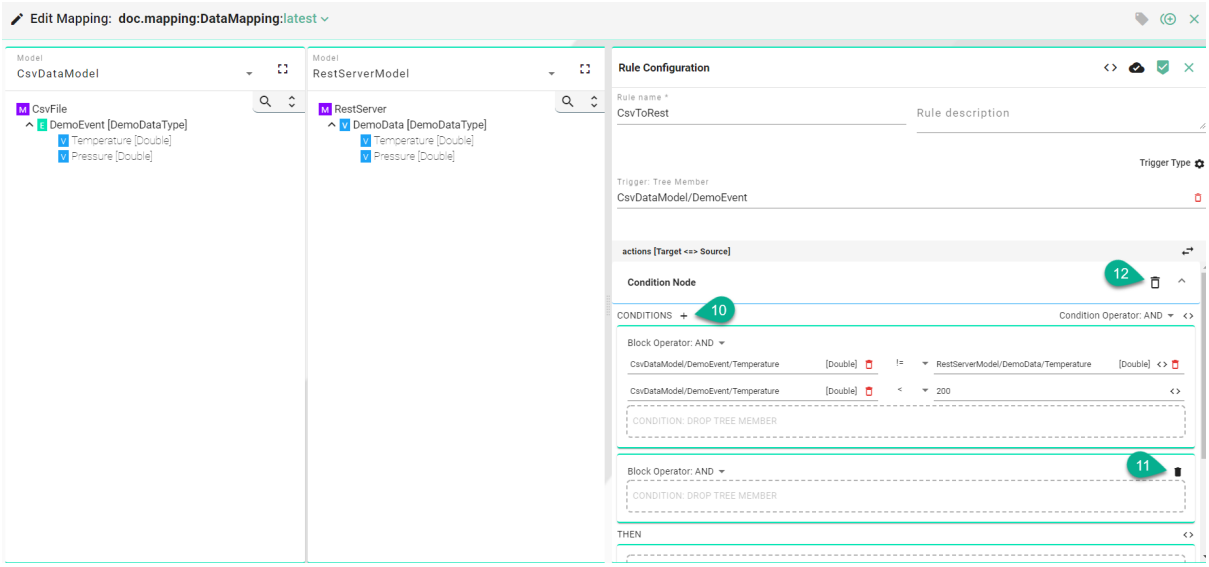
- Click on the "Add condition block" button **(1)**.
- Drag and drop a tree member to build up the condition **(2)** and **(3)** or use the Literal Node button to enter a custom value.
- Select a condition operator **(4)**. The following operators are available: ==, !=, <, <=, >, >=.



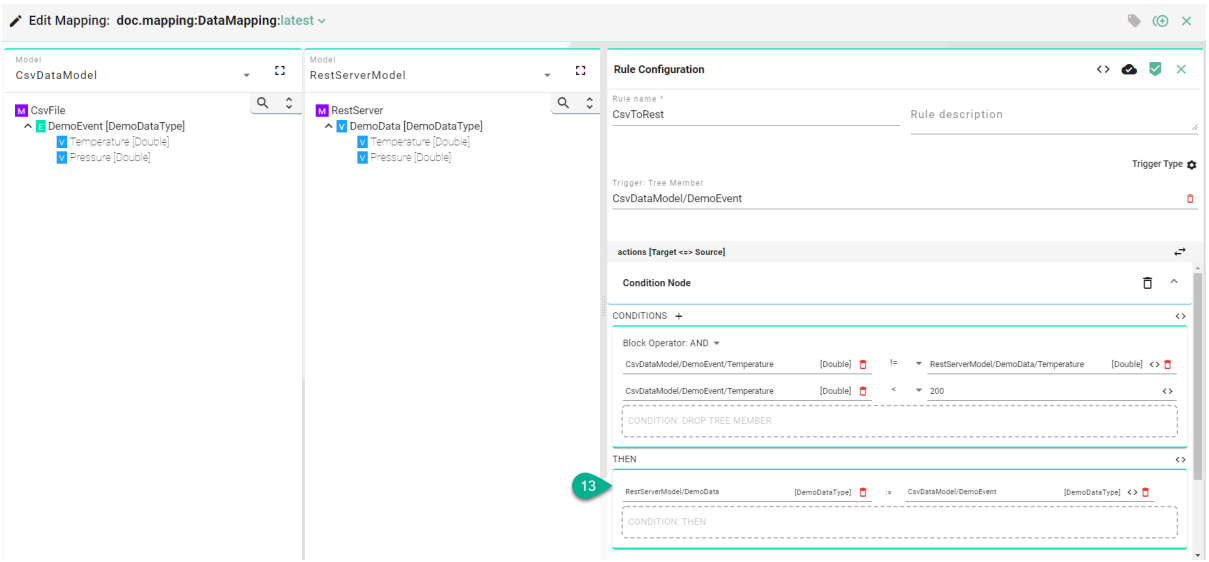
- To add multiple conditions **(5)** select the block operator **(6)**.
- Click on the "Literal Node" button **(7)** to enter a custom value **(8)**.
- Select a condition operator **(9)**.



- Click on the "Add Condition Block" button **(10)** to add a new one.
- Click on the "Delete Condition Block" button **(11)** to remove a condition block and select the "Delete" button **(12)** to remove the condition.

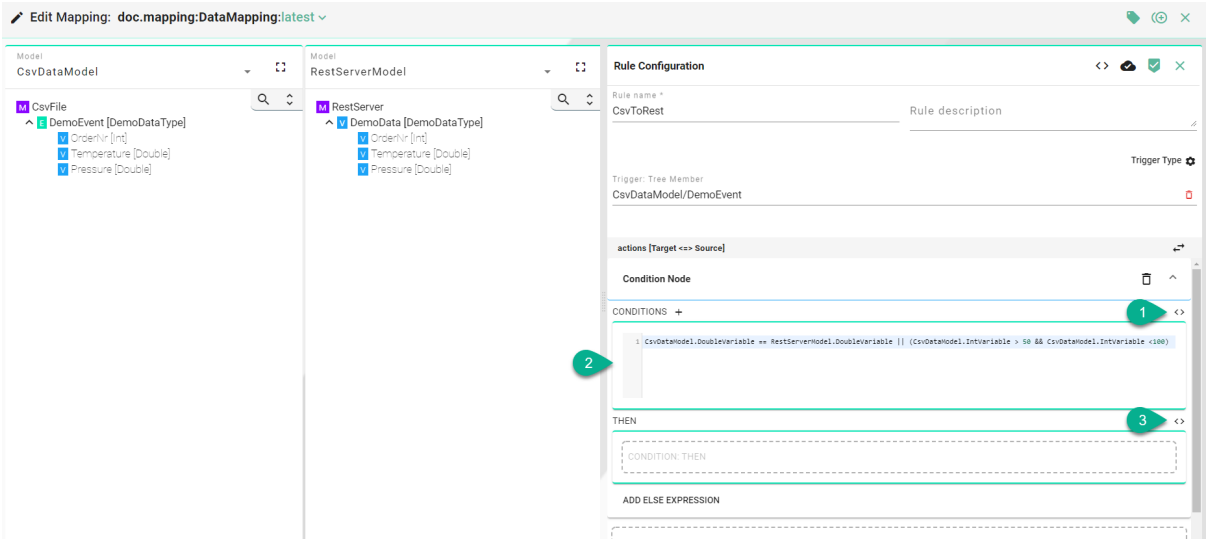


In the "THEN" **(13)** section, drag and drop the Target and Source Information Model nodes, using either the *simple* or the *complex* assignment methods.



Actions with Custom Conditions

- Click on the "Source Code" button **(1)**.
- Input code for a complex condition **(2)**.



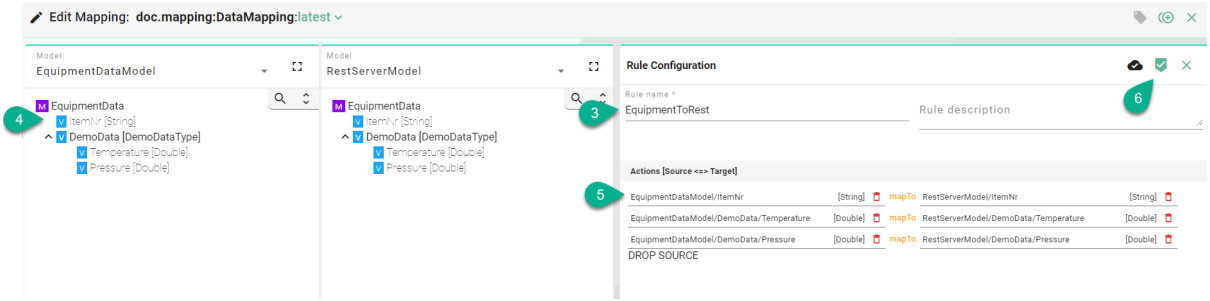
- For the "THEN" section, either use drag and drop to add the target and source Information Model nodes, or click the "Source Code" button **(3)** to input code.

Multi Rule

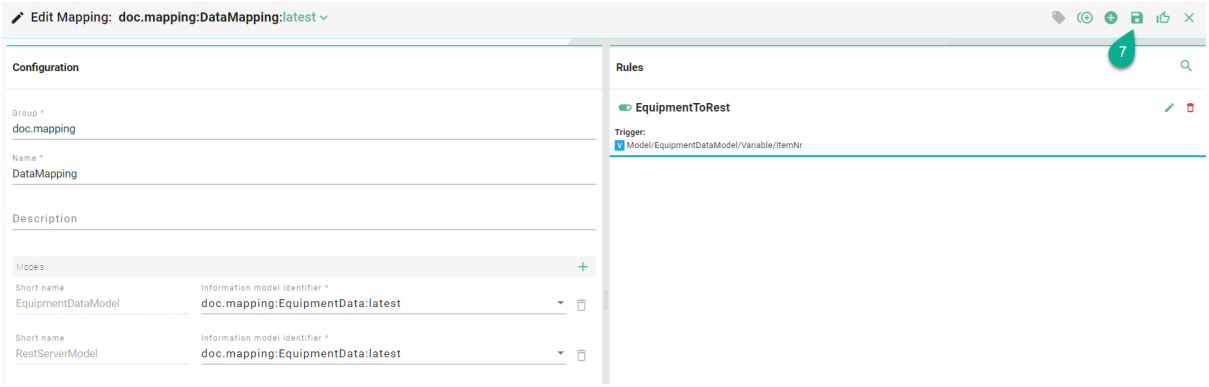
The following screenshot displays the Multi Rule Editor. Here, no specific trigger can be set; instead, each source element from an Information Model behaves as a trigger. When a source element is updated, the assignment to the target element is performed.

> **Note**
>
> The Multi Rule configuration treats each source as a trigger.

- Enter "Rule name" **(3)**.
- Drag and drop the *Source* Information Model nodes **(4)** one by one into the Source field **(5)**.
- The Source and Target information must match on a one-to-one basis (e.g., String to String). Allowed nodes for Source and Target include Simple Variables and Variables from a Complex Variable.
- After all mandatory fields are filled out, click the "Apply" button **(6)** to save the newly created rule.

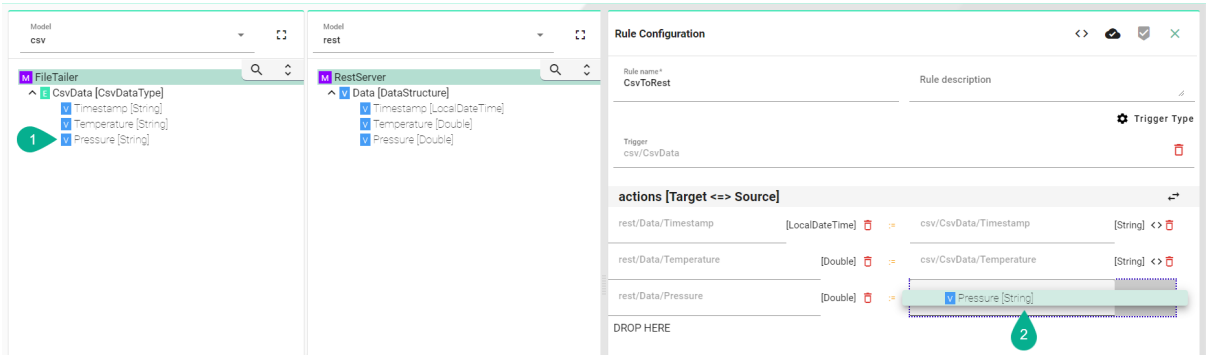


- The Multi Rule Editor closes, and the newly created rule appears in the Rules List.
- Click the "Save" button **(7)** located in the upper right corner to save the mapping.



Implicit Type Conversion

Within the graphical mapping, *Variables* and *Properties* of different data types are implicitly converted. For example, a variable of type String **(1)** can be assigned to a variable of type Double **(2)** without the need for manual data type conversion through code entry.



The table below outlines the implicit conversions supported:

Data Type	Convertible To/From
String	Char, Byte, Short, Int, Long, Float, Double, LocalDateTime, OffsetDateTime, Boolean
Long	LocalDateTime, OffsetDateTime
LocalDateTime	OffsetDateTime

Code-based Rules

The main target of SMART**UNIFIER** is to build up the connectivity between systems. Sometimes integrations become more complex and it might require to build up Rules via the code editor using the Scala programming language. SMART**UNIFIER** extends the Scala programming language with addition operators and methods to simplify the realization of data transfer between Information Models.

Similar to Mappings via drag and drop, there is no knowledge of the underlying communication protocol (e.g., MQTT, OPCUA, etc.) needed. Protocols are hidden behind the corresponding Information Models. The parameter values of an Information Model are stored in the objects of type *VariableDefinition[T]* or *PropertyDefinition[T]*. These contain additional information and methods rather than just the parameter values. They also provide methods to listen for changes and conversion between variable types.

Basics

Rule construct

A rule always starts with a *Trigger* **(1)**. The trigger can represent an element of the Information Model, such as a *Variable*, *Event*, *Command*, or it can be time-based.

After the trigger, call mapTo **(2)** and define the function body by adding curly braces **(3)**.

Depending on the trigger, declare the TriggerInstance **(4)**. Use naming that corresponds to the type of the trigger.

```
Trigger  mapTo  {  TriggerData  =>

}
```

The *Source* **(5)** is the content of the TriggerInstance. For example, if the trigger is a Variable, then the Source is an instance of that Variable.

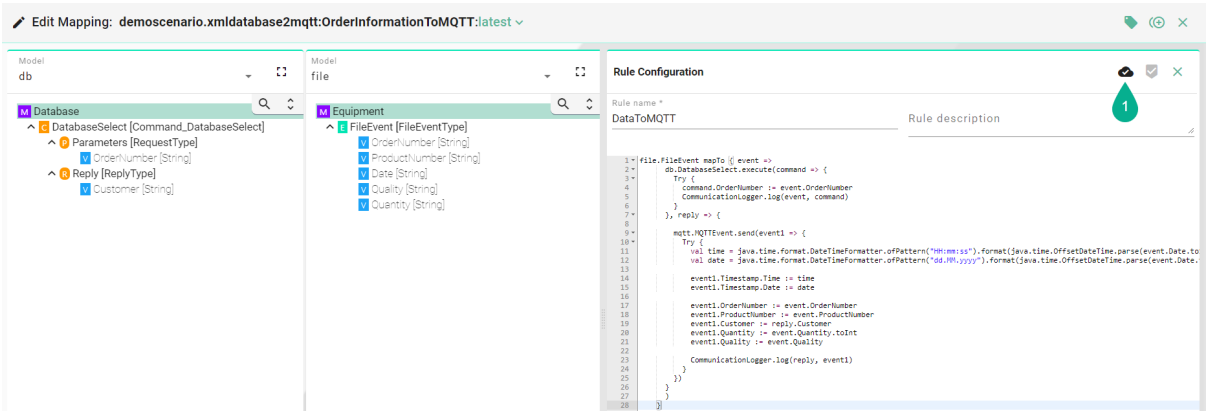
To assign the Source to the *Target*, use the := operator **(6)**.

The Target can be any variable you want to map to **(7)**.

```
Trigger  mapTo  {  TriggerData  =>
         Target  :=  Source

}
```

Compiling

Compile the code for the selected rule by clicking the "Compile" button **(1)** and check for compilation errors before saving the rule.



Logging

Logging can be added in the Rule implementation by calling - **CommunicationLogger.log** (line 5)

Listing 1: Rule with Logging

```
1 EquipmentModel.Alarm mapTo {variable =>
2    MesModel.EquipmentAlarm.send(event => {
3      Try {
4        event.EquipmentId := EnterpriseModel.EquipmentName
5        CommunicationLogger.log(variable, event)
6      }
7    })
8 }
```

Trigger Types

Tree Member

The following Information Model elements can be used as a trigger: *Variables*, *Events*, *Commands*. The snippet below shows how the trigger is defined:

```
<Information Model>.<Element from the Information Model> mapTo { <Element type> =>
```

Listing 2: Tree Member as Trigger

```
1 EquipmentDataModel.ItemNr mapTo { variable =>
2    Try {
3      EquipmentDataModel.DemoData.Temperature := RestServerModel.DemoData.
  ↪Temperature
4      EquipmentDataModel.DemoData.Pressure := RestServerModel.DemoData.Pressure
5    }
6  }
```

Schedulers

With schedulers, you can execute *Rules* at specified times or intervals. You can choose from the following scheduler types:

- *Fixed Rate Scheduler*
- *Fixed Delay Scheduler*

Typically, the scheduler is started automatically and executes the rule once the instance is started, and used channels are in the "Connected" state.

However, you can manually trigger the execution and termination of a rule by using the start/stop function of the scheduler:

Listing 3: Start Rule

```
_trigger.Schedulers("<name of rule>").start()
```

Listing 4: Stop Rule

```
_trigger.Schedulers("<name of rule>").stop()
```

Fixed Rate Scheduler

Rules can be scheduled to run continuously at a fixed rate. Instead of defining an element of the Information Model as a trigger, the **fixedRateScheduler** method can be used. The snippet below shows how the fixed rate scheduler is defined:

```
_trigger.fixedRateScheduler(<Cron Expression>)
```

Listing 5: Fixed Rate Scheduler

```
1 _trigger.fixedRateScheduler("0/1 * * * * ? *") mapTo(() => {
2   model1.StringVariable := model2.StringVariable
3 })
```

Example expressions

Expression	Description
`0/1 * * * * ?`	Every second
`0/20 * * * * ?`	Every 20 seconds
`15 0/2 * * * ?`	every other minute, starting at 15 seconds past the minute.
`0 0/2 8-17 * * ?`	every other minute, between 8am and 5pm (17 o'clock).
`0 0/3 17-23 * * ?`	every three minutes but only between 5pm and 11pm
`0 0 10am 1,15 * ?`	10am on the 1st and 15th days of the month
`0,30 * * ? * MON-FRI`	every 30 seconds on Weekdays (Monday through Friday)
`0,30 * * ? * SAT,SUN`	every 30 seconds on Weekends (Saturday and Sunday)

Fixed Delay Scheduler

Rules can be scheduled to run at a fixed rate with an initial delay. The snippet below shows how the fixed delay scheduler is defined:

```
_trigger.fixedDelayScheduler(<Initial Delay>, <Period>, <Unit>) mapTo(() =>
```

Listing 6: Fixed Delay Scheduler

```
1 _trigger.fixedDelayScheduler(10, 60, SECONDS) mapTo(() => Try{
2   EquipmentDataModel.DemoData.Temperature := RestServerModel.DemoData.
  ↪Temperature
3   EquipmentDataModel.DemoData.Pressure := RestServerModel.DemoData.Pressure
4 })
```

Timeout Scheduler

Rules can be scheduled to run after a specific timeout. The snippet below demonstrates how the timeout scheduler is defined:

```
_trigger.timeoutScheduler(<Delay>, <Unit>) mapTo(() =>
```

Listing 7: Timeout Scheduler

```
1 _trigger.timeoutScheduler(60, SECONDS) mapTo(() => Try{
2   EquipmentDataModel.DemoData.Temperature := RestServerModel.DemoData.
  ↪Temperature
3   EquipmentDataModel.DemoData.Pressure := RestServerModel.DemoData.Pressure
4 })
```

Target-to-Source Mapping

Node Types Sharing the Same Custom Data Type

When the target and source *Node Types* in the Information Model are both of the same *Custom Data Type*, the Mapping can be simplified:

Listing 8: Mapping of two Events with the same type

```
event1 := event2
```

The two Node Types have to be of the same kind e.g., both are Events or both are Variables.

Node Types with different Custom Data Type

The examples demonstrate how to map values between source and target variables.

Variables to Events

This mapping is utilized when static data needs to be transformed into an Event. This is often the case when data originates from a variable-based data server (such as OPC UA server, Modbus, Iso-On-TCP) and is required to be mapped to an event or message-based target system (like MQTT, Kafka, Databases, etc.).

The example below illustrates the mapping of variables from the **EnterpriseModel** and the **EquipmentModel** to an Event within the **MesModel**:

- Trigger: **EquipmentModel.Alarm** (line 1)
- TriggerInstance of EquipmentModel.Alarm: **variable** (line 1)

- Invoke the send method on the **EquipmentAlarm** Event (line 2) and define the TriggerInstance as **event** (line 2)
- Variable assignment is performed using the assignment operator **:=**. Both target and source are specified by entering the path of the variables in the Information Model, for example, **event.EquipmentId** and **EnterpriseModel.EquipmentName** (line 4)

Listing 9: Rule - StartOrder - Variable/Event

```
1  EquipmentModel.Alarm mapTo {variable =>
2     MesModel.EquipmentAlarm.send(event => {
3       Try {
4         event.EquipmentId := EnterpriseModel.EquipmentName
5         event.OrderNr := EquipmentModel.CurrentOrder.OrderNr
6         event.MaterialID := EquipmentModel.CurrentMaterialID
7         event.AlarmInfo := EquipmentModel.AlarmInfo
8         CommunicationLogger.log(variable, event)
9       }
10    })
11 }
```

Event to Variables

This mapping is utilized when dealing with event-driven data that needs to be mapped to variables. This scenario often occurs when data originates from an event or message-based system (e.g., MQTT, Kafka, Databases, etc.) and needs to be mapped to a variable-based data server (such as OPC UA server, Modbus, Iso-On-TCP).

The example below outlines the mapping of values from the **TransferNewOrder** Event in the **MesModel** into variables within the **EquipmentModel**:

- The Trigger is specified by entering the path of the Event **MesModel.TransferNewOrder** (line 1). Since an Event is utilized as the Trigger, the TriggerInstance is appropriately named **event** (line 1).
- In the function body, the Complex Variable **NewOrder** and the Simple Variable **NewMESOrderFlag** are provided with data from the MesModel's **TransferNewOrder** Event.
- Targets are specified by entering the path of the variables, such as **EquipmentModel.NewOrder.OrderNr** (line 3).
- To assign values to **OrderNr**, **MaterialNr** and **Quantity** of the Complex Variable **NewOrder**, enter the TriggerInstance event followed by the variable name from the TransferNewOrder Event, e.g., **event.OrderNr** (line 3).
- In this case it is also possible to assign the variable **NewMesOrderFlag** a Boolean value like **true** (line 6)

Listing 10: Rule - TransferNewOrder - Event/Variable

```
1  MesModel.TransferNewOrder mapTo { event =>
2     Try {
3       EquipmentModel.NewOrder.OrderNr := event.OrderNr
4       EquipmentModel.NewOrder.MaterialNr := event.MaterialNr
```

(continues on next page)

(continued from previous page)

```
    EquipmentModel.NewOrder.Quantity := event.Quantity
    EquipmentModel.NewMESOrderFlag := true
  }
}
```

Event to Commands

This mapping is employed when dealing with event-driven data that needs to be mapped to a Command. This scenario may arise when incoming event or message-driven data should be enriched with data from another system (such as a database or a REST server) before being further mapped to another event-driven message.

The following scenario describes a rule that maps incoming data from a file to MQTT. When the FileEvent is triggered, the rule first executes the **DatabaseCommand** to retrieve data from a database (the result of the reply can be accessed directly afterward):

- Trigger is specified by entering the path of the Event **file.FileEvent** (line 1). Since an Event serves as the Trigger, the TriggerInstance should be named **event** (line 1)
- Within the function body, execute a Command. The execution of a Command is specified by entering the path of the Command and calling the **execute** function at the end of the path (line 2). The TriggerInstance is named **command** (line 4).
- Lines 4-6 illustrate the first part of the Command execution, where values from the source model are assigned to the Command Parameters.
- Every Command includes a Reply, which necessitates defining the reply section (line 8).
- After retrieving data from the database, send out the data over MQTT. In the reply function body, specify the path of the **MqttEvent**. Since this is the second Event, the TriggerInstance can be named **event1** (line 10).
- Within the reply function body, assign values from the **FileEvent** (lines 11-13) as well as from the Reply (lines 14-15) to the **MqttEvent**.

Listing 11: Rule - File2MqttWithDB - Event/Commands

```
file.FileEvent mapTo {event =>
  database.DatabaseCommand.execute(command => {
    Try {
      command.orderNr := event.orderNr
      command.materialNr := event.materialNr
      CommunicationLogger.log(event, command)
    }
  }, reply => {
    mqtt.MqttEvent.send(event1 => {
      Try {
        event1.Quality := event.quality
        event1.OrderNr := event.orderNr
        event1.MaterialNr := event.materialNr
        event1.Customer := reply.customer
        event1.Product := reply.product
        CommunicationLogger.log(reply, event1)
```

(continues on next page)

(continued from previous page)

```
        }
    })
  })
}
```

Properties to Variables

When a *Property* serves as the source and a Variable as the target, the mapping is straightforward: the Property is assigned to the Variable using the assignment operator :=. This approach may be utilized when dealing with an XML structure that includes XML-Attributes, which are modeled as Properties in the Information Model, while the target system expects the data to be presented as Variables.

Listing 12: Rule - Property/Variables

```
propertyNodeType := variableNodeType
```

Mapping including Lists

If there are *Lists* structures within an Information Model that need to be mapped to another Information Model, it is necessary to iterate through the list items using a foreach loop.

The following scenario describes a Rule that maps incoming data from a file to MQTT. The MQTT Model contains a List called **DataList**.

- Initialize a variable named **listItem** reference a **newItem** in the **DataList** (line 6)
- Then, assign the value from the file event to this variable listItem (line 8)

Listing 13: Rule - FileToMQTT - Lists

```
csv.FileEvent mapTo { event =>

    event.items.foreach { item =>
      mqtt.MqttEvent.send(event1 => {
        Try {
          val listItem = event1.DataList.newItem

          listItem.Timestamp := item.Timestamp
          listItem.Pressure := item.Alarmlevel

          CommunicationLogger.log(event, event1)
        }
      })
    }
}
```

Note

Lists can only be mapped in the code view.

SMARTUNIFIER Code Constructs

Rules are written in the Scala programming language. SMART**UNIFIER** also includes custom code constructs that can be used within mappings, allowing for operations such as type conversions directly at the variable level.

Converters

If the variables to be mapped to each other are not of the same data type, use the provided type converters. Converters can be used on Information Model nodes such as *Variables* and on *Properties* .

Method	Description	Example
toBoolean(definition: TVariableDefinition[T])	Converts a variable to a Boolean	toBoolean(m1.IntVariable)
toBoolean(definition: TPropertyDefinition [T])	Either the literal true or the literal false	"
toByte (definition: TVariableDefinition[T])	Conversion of a variable to an Byte	toByte(m1.IntVariable)
toByte (definition: TPropertyDefinition [T])	8 bit signed value. Range from -128 to 127	"
toShort (definition: TVariableDefinition[T])	Conversion of a variable to a Short	toShort(m1.IntVariable)
toShort (definition: TPropertyDefinition [T])	16 bit signed value. Range -32768 to 32767	"
toInt (definition: TVariableDefinition[T])	Conversion of a variable to an Integer	toInt(m1.StringVariable)
toInt (definition: TPropertyDefinition [T])	32 bit signed value. Range -2147483648 to 2147483647	"
toLong (definition: TVariableDefinition[T])	Conversion of a variable to a Long	toLong(m1.IntVariable)
toLong (definition: TPropertyDefinition [T])	64 bit signed value. Range -9223372036854775808 to 9223372036854775807	"
toFloat(definition: TVariableDefinition[T])	Conversion of a variable to a Float	toFloat(m1.IntVariable)
toFloat (definition: TPropertyDefinition [T])	32 bit IEEE 754 single-precision float	"
toDouble(definition: TVariableDefinition[T])	Conversion of a variable to a Double	toDouble(m1.IntVariable)
toDouble (definition: TPropertyDefinition [T])	64 bit IEEE 754 double-precision float	"
toStr(definition: TVariableDefinition[T])	Conversion of a variable to a String	toStr(m1.IntVariable)
toStr (definition: TPropertyDefinition [T])	A sequence of Chars	"

Math Operators

Math Operator methods can be utilized to perform calculations, such as addition, subtraction, multiplication, and division. If there's a need to perform calculations on the values of a variable within the mapping before sending data to the target system, the following methods can be employed:

Method	Description	Example
add(Option[T],Double)	Addition of a variable with a numeric data type and a Double value	add(model.IntVariable, 2)
sub(Option[T],Double)	Subtraction of a variable with a numeric data type and a Double value	sub(model.IntVariable, 2.5)
mult(Option[T],Double)	Multiplication of a variable with a numeric data type and a Double value	mult(model.IntVariable, 3)
div(Option[T],Double)	Division of a variable with a numeric data type and a Double value	div(model.IntVariable, 3.5)

String Operators

String Operator methods can be utilized to perform String manipulation:

Method	Description	Examples
concat(variable, variable)	Concatenates two strings together.	concat(myModel.myStringVariable, "World") concat("Hello", myModel.myStringVariable) concat(myModel.myStringVariableA, myModel.myStringVariableB)
contains(variable, sequence)	Checks and returns true if and only if this string contains the specified sequence of char values.	contains(myModel.myStringVariable, "Hello") contains(myModel.myStringVariable, myModel.myOtherStringVariable)
matches(variable, regex)	Checks and return true if the string matches the given regular expression.	matches(myModel.myStringVariable, "Hello.*") matches(myModel.myStringVariable, myModel.myRegExStringVariable)
replace(variable, target, replacement)	Replaces first occurrence of a substring within the string with the specified replacement.	replace(myModel.myStringVariable, "T", "X") replace(myModel.myStringVariable, myModel.search, "X") replace(myModel.myStringVariable, myModel.search, myModel.replace)
replaceAll(variable, target, replacement)	Replaces all occurrences of a substring within the string with the specified replacement.	replaceAll(myModel.myStringVariable, "T", "X") replaceAll(myModel.myStringVariable, myModel.search, "X") replaceAll(myModel.myStringVariable, myModel.search, myModel.replace)
substring(variable, beginIndex)	Extracts a string that is a substring of this string. The substring begins with	substring(myModel.myStringVariable, 3)

Helpers

Helpers are methods that can be used to simplify the mapping process. They can be used to compare the value of a variable with a given value, or to map child variables from one complex variable to another.

> **Hint**
>
> It is possible to convert a String to a Local- and OffsetDateTime without any extra parsing.

Method	Description	Example
equals(variable, variable)	Compares the value of a variable with a given value	equals(myModel.myStringVariable, "Foo")
mapAndAssignChildren(complexVariable1, complexVariable2)	Maps child variables from one complex variable to another mapAndAssignChildren(<source>, <target>). Variables must be of the same type and have the same id	mapAndAssignChildren(m1.ComplexVariableDepth1, m2.ComplexVariableDepth1)

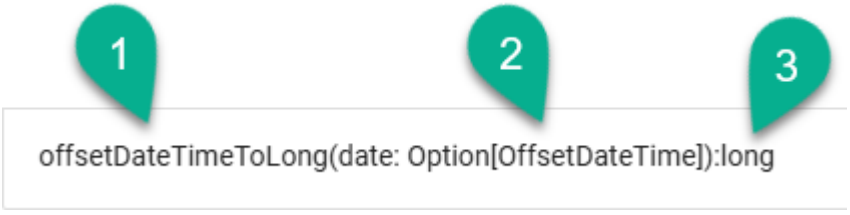
Time conversions

> **Note**
>
> When dealing with OffsetDateTime, Zoneoffset.UTC is set as default.

> **Note**
>
> When formatting the following formats are set als defaults: **ISO_DATE_TIME** = "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'", **ISO_DATE** = "yyyy-MM-dd", **ISO_TIME** = "HH:mm:ss.SSS'Z'"

Method structure explanation

- **(1)** Method name
- **(2)** Input type
- **(3)** Output type



Method	Description	Example
longToOffsetDateTime(timestamp: Option[long], offset: ZoneOffset = ZoneOffset.UTC):OffsetDateTime	Parses a Unix timestamp provided in a long variable to OffsetDateTime	longToOffsetDateTime(Option(1732873920000)) --> java.time.OffsetDateTime longToOffsetDateTime(myModel.MyUnixTimestamp) --> java.time.OffsetDateTime
longToLocalDateTime(timestamp: Option[long], offset: ZoneOffset = ZoneOffset.UTC):LocalDateTime	Parses a Unix timestamp provided in a long variable to LocalDateTime	longToLocalDateTime(Option(1601536800000)) --> java.time.LocalDateTime longToLocalDateTime(myModel.MyUnixTimestamp) --> java.time.LocalDateTime
parseOffsetDateTime(dateTime: Option[String], formatter: DateTimeFormatter = DateTimeFormatter.ISO_OFFSET_DATE_TIME):O	Parses a String to OffsetDateTime	parseOffsetDateTime(Option("2024-11-29T08:32:00+01:00")) --> java.time.OffsetDateTime
parseLocalDateTime(dateTime: Option[String], formatter: DateTimeFormatter = DateTimeFormatter.ISO_LOCAL_DATE_TIME):Lo	Parses a String to LocalDateTime	parseLocalDateTime(Option("2024-12-13T10:15:30")) --> java.time.LocalDateTime
formatOffsetDateTime(dateTime: Option[java.time.OffsetDateTime], formatter: DateTimeFormatter = DateTimeFormatter.ISO_OFFSET_DATE_TIME):St	Formats an OffsetDateTime	formatOffsetDateTime(Option(java.time.OffsetDateTime)) --> "2024-11-29T08:32:00+01:00"
formatLocalDateTime(dateTime: Option[java.time.LocalDateTime], formatter: DateTimeFormatter = DateTimeFormatter.ISO_LOCAL_DATE_TIME):Str	Formats a LocalDateTime	formatLocalDateTime(Option(java.time.LocalDateTime)) --> "2024-12-13T10:15:30"
localDateTimeToOffsetDateTime(dateTime: Option[LocalDateTime], offset: ZoneOffset = ZoneOffset.UTC):OffsetDateTime	Parses a LocalDateTime to OffsetDateTime	localDateTimeToOffsetDateTime(Option(LocalDateTime)) --> java.time.OffsetDateTime
localDateTimeToLong(dateTime: Option[LocalDateTime], offset: ZoneOffset = ZoneOffset.UTC):long	Parses a LocalDateTime to long	localDateTimeToLong(Option(LocalDateTime)) --> 1601536800000
offsetDateTimeToLocalDateTime(dateTime: Option[OffsetDateTime]):LocalDat	Parses an OffsetDateTime to LocalDateTime	offsetDateTimeToLocalDateTime(Option(OffsetDateTime)) --> 2024-12-13T10:15:30
offsetDateTimeToLong(date: Option[OffsetDateTime]):long	Parses an OffsetDateTime to Long	offsetDateTimeToLong(Option(OffsetDateTime)) --> 1732873920000

Here are some example of some Date conversions:

> **Note**
>
> When converting a String to a Local-/OffsetDateTime, no parsing needs to be done manually, as its done automatically in the mapping.

Method	Input	Output
longToOffsetDateTime	1601536800000	2024-12-13T10:15:30+01:00
StringToLocalDateTime	"2024-12-13T10:15:30"	2024-12-13T10:15:30
StringToOffsetDateTime	"2024-11-29T08:32:00+01:00"	2024-11-29T08:32:00+01:00

During mapping, create a new Rule, switch to the code view **<>** and follow the following concept:

```
SourceModel.SourceEvent/ComplexVar mapTo { event/variable =>
  Try {

   TargetModel.TargetEvent/ComplexVar := parsing method(event/variable.
↪SourceVar)

  }
}
```

```
Equipment.DateTimeConversion mapTo { event =>
  Try {
    target.complexVar.Local2Long := localDateTimeToLong(event.Local2Long)
    target.complexVar.Long2Local := longToLocalDateTime(event.Long2Local)
    target.complexVar.String2Local := parseLocalDateTime(event.String2Local)
    target.complexVar.FormatLocal := formatLocalDateTime(event.FormatLocal)
    target.complexVar.Offset2Local := offsetDateTimeToLocalDateTime(event.
↪Offset2Local)
  }
}
```

Loops (foreach)

In some use cases, it may be necessary to iterate through a collection if the Information Model contains a list or an array. In this case, call the items method on the list element of the Information Model, followed by foreach (line 13).

Listing 14: Code constructs - Loops

```
1 import java.time.LocalDateTime
2 import java.time.Instant
3 import java.time.ZoneId
4
5 equipment.FileEvent mapTo { event =>
6  val newImportDate = LocalDateTime.ofInstant(Instant.ofEpochMilli(System.
```

(continues on next page)

(continued from previous page)

```
↪currentTimeMillis()), ZoneId.systemDefault())
 db.MainDatabaseEvent.send(event1 => Try {
   event1.ImportDateTime := newImportDate
   event1.StepId := event.StepId

   event.analysisData.items.foreach {

     case analysisDataType:␣
↪ComplexCollectionVariableDefinition[AnalysisDataType] => {

       val analysisDataItem = event1.analysisDataTable.newItem

       analysisDataItem.name := analysisDataType.name
       analysisDataItem.length := analysisDataType.length
     }
   }
 })
}
```

Conditions (If - statements)

Within a rule, it's possible to implement conditions using Scala's conditional expressions. **If** statements can be used to test a condition before executing the subsequent block. For example, this can be utilized to check if a certain condition is met before executing an event (line 3).

Listing 15: Code constructs - Conditions

```
equipment.ActiveOrder.State mapTo { variable =>
 logger.info(s"Active order state: ${variable.value} - Processing Finished")
 if (variable.value == 3) {
   mes.NotifyOrderFinished.send(event => {
     Try{
       event.EquipmentId := equipment.EquipmentInformation.EquipmentType
       event.OrderNr := equipment.ActiveOrder.OrderInformation.OrderNo
       event.ProductNumber := equipment.ActiveOrder.OrderInformation.ProductNo
       event.QuantityOk := equipment.ActiveOrder.QuantityOk
       event.QuantityNOk := equipment.ActiveOrder.QuantityNOk
     }
   })
 }
}
```

Exception Handling (Try/Catch)

Exception Handling is an integral part of the SMART**UNIFIER** mapping logic. The **mapTo** and **send** callbacks expect a return value of the Scala type **Try**. If an exception occurs in one of the rules, SMART**UNIFIER** logs the exception and displays a notification in the manager. Supported Communication Channels take further actions once an exception has occurred. For example, the File Reader Channel moves a file that initially triggered a rule into an error folder.

In the example below, a **Try** block is placed after each command and event call (lines 2 and 4).

Listing 16: Code constructs - Exception Handling

```
1  database.update mapTo { (updateCommand, reply) =>
2   Try {
3     api.AmorphAPI.send(event =>
4       Try {
5
6         event.id := updateCommand.Identifier.Id
7         event.name := updateCommand.Identifier.Name
8
9         updateCommand.Status.items.foreach(item => {
10          val statusItem = event.status.newItem
11          statusItem.index := item.Index
12          statusItem.value := itemValue
13        })
14      }
15    )
16  }
17 }
```

Breaking out of Rules

You can break out of a rule by calling the **Break()** method in your code. Any code defined after the **Break()** method will not be executed.

Example 1: The **Break()** method can be used to stop the execution of the code if, for example, a variable value is not present (lines 2-4) but is needed later (line 9).

Listing 17: Code constructs - Break()

```
1  def rule_r1(): Unit = {
2    if(model1.myVariable.isEmpty()) {
3      Break()
4    }
5
6   //... mapping code here ...
7
8   model1.myVariable.mapTo {
9     variable => model2.myVariable := variable
10  }
11 }
```

Example 2: Breaking out of a loop if the iterator does not have a next element (line 4) and calling **Break()** (line 5).

Listing 18: Code constructs - Break()

```
1  def rule_r2(): Unit = {
2    val iterator = model1.myList.items.iterator
3    while(iterator.hasNext){
```

(continues on next page)

(continued from previous page)

```
    if(iterator.next().isEmpty()){
      Break()
    }
  }
   // ... mapping code here ...
}
```

Device Types

What are Device Types

The SMART**UNIFIER** Device Type acts as a template for a Communication Instance that can be reused (i.e., one instance represents one equipment). Multiple Communication Instances, which share common configuration parameters, can be created based on one Device Type.

In the simplest integration scenario of a single equipment, one Device Type must be created to create a Communication Instance. This enables to create and spin up two Instances for test and production operation.

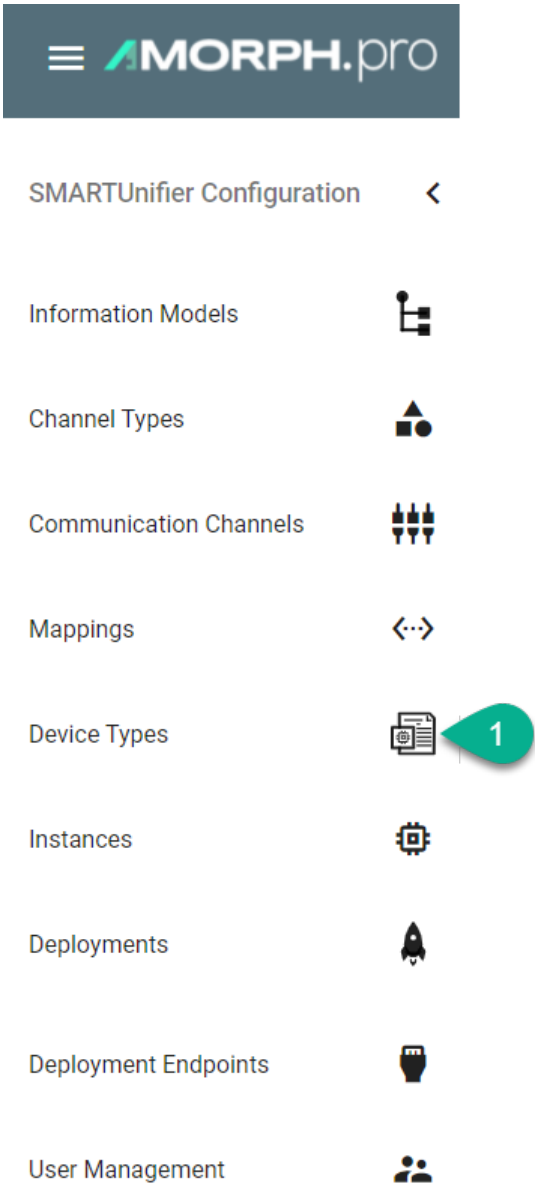
A Device Type itself contains one or multiple Mappings, which allows to build up communication flows between multiple systems (how this can be done is shown in the SMARTUNIFIER Demonstrator). Each Mapping contains one or multiple Information Models with its associated Communication Channel.

Device Types are especially important, when integrating several similar pieces of equipment or devices.

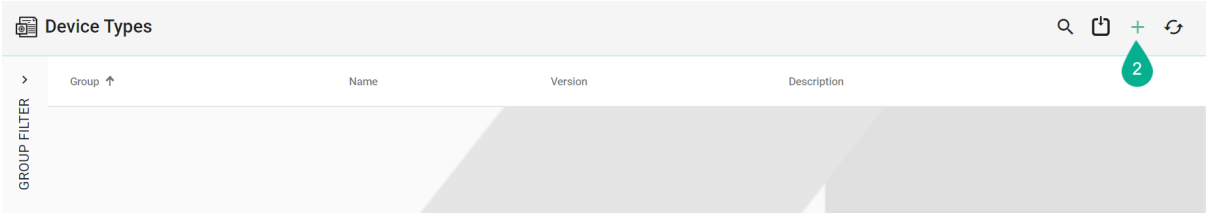
How to create a new Device Type

Follow the steps described below to create a SMART**UNIFIER** Device Type.

- Select the SMART**UNIFIER** Device Type perspective **(1)**.



- Click on the "Add Device Type" button in the upper right corner **(2)**.



- The creation of a Device Type is divided into two parts. First, provide basic information about the Device Type, such as its Group, Name, and Version. Additionally, you can provide a short description, if desired **(3)**.
- In the next step, provide one or multiple previously created *Mappings*. To do so,
  - Click the "Add Mapping" button **(4)**
  - After selecting a Mapping **(5)**, the associated Information Models will appear

  - If the wrong Mapping was selected, click the "Delete Mapping" button to remove it from the Device Type **(6)**
  - Finally, select a *Communication Channel* for each *Information Model* from the drop-down **(7)**

- To save the new Device Type, click the "Save" button located in the top right corner of the screen **(8)**.

Communication Instances

What are Instances

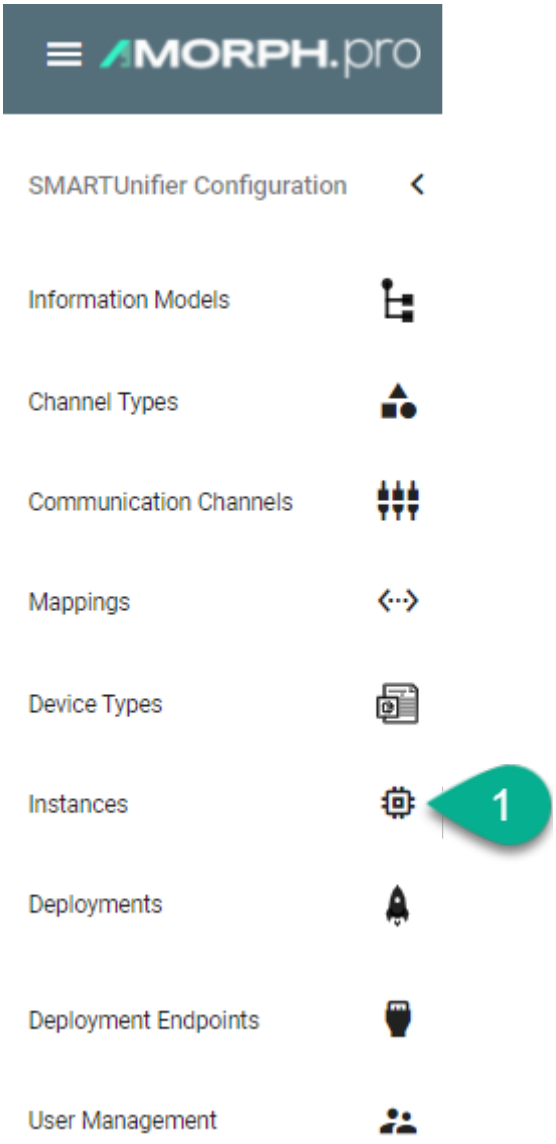
A SMART**UNIFIER** Instance is a dynamically created application that can be deployed to any suitable IT resource (e.g., Equipment PC, Server, Cloud), and which provides the connectivity functionality configured. Therefore, a SMART**UNIFIER** Instance uses one or multiple Mappings and selected Communication Channels from a previously defined *Device Type*.

The Communication Instance is the final component and when deployed it acts as a standalone application that provides the connectivity between two or multiple systems. The granular configuration of the Communication Channels can be done on this level (e.g., changing the IP address) which is useful when there are multiple Communication Instances created based on the same Device Type.

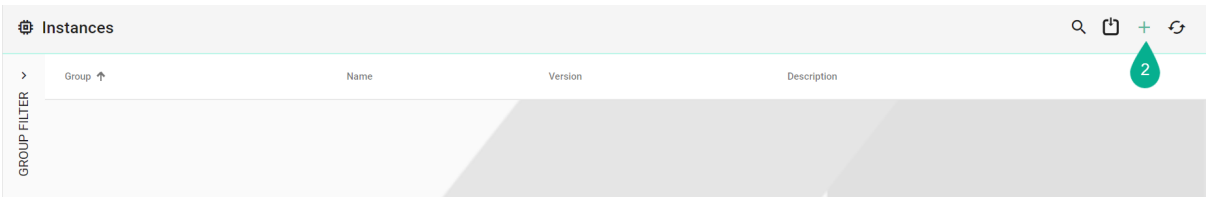
How to create a new Instance

Follow the steps described below to create a SMART**UNIFIER** Instance.

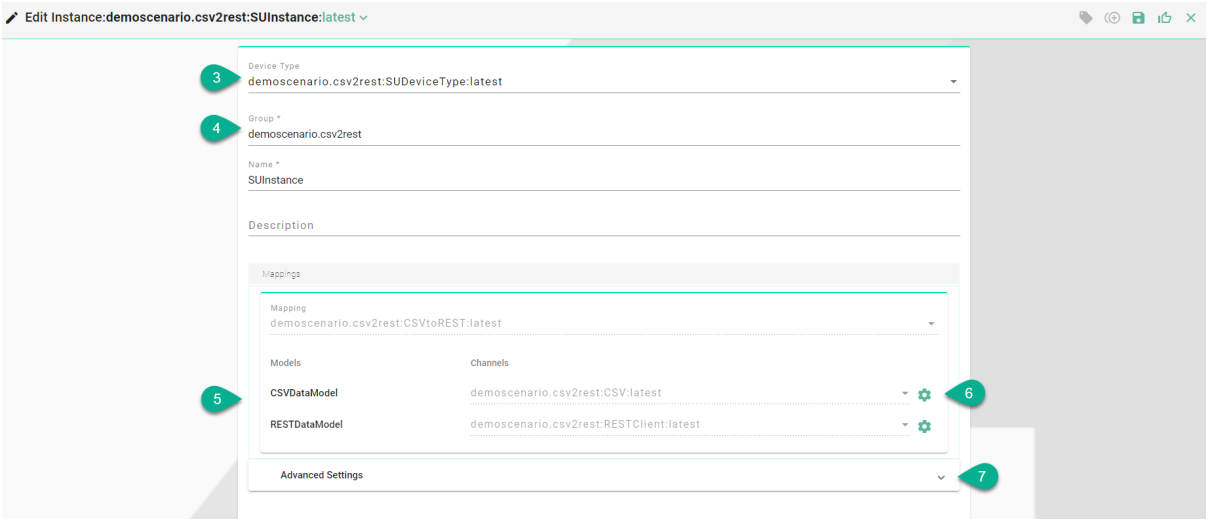
- Select the SMART**UNIFIER** Instances perspective **(1)**



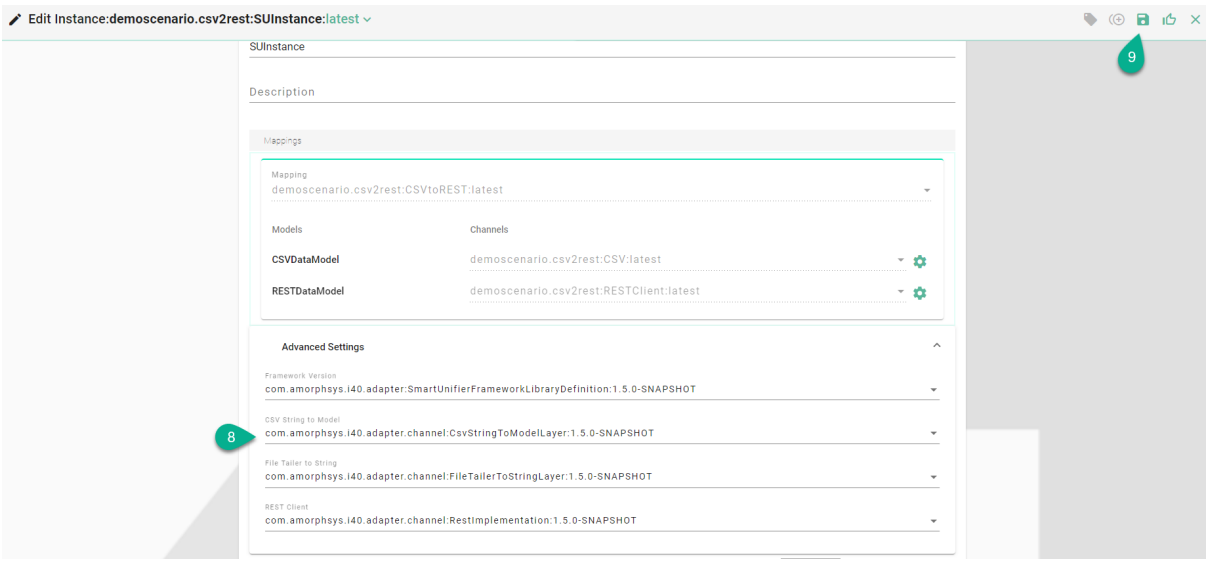
- Click on the "Add Instance" button in the upper right corner **(2)**



- Select a Device Type from the drop-down **(3)**
- The instance details are automatically populated based on the Device Type **(4)**, but you can still modify the Group, Name, Version, and Description fields as needed
- Mappings defined in the Device Type will appear in the Mapping area **(5)**
- To change the existing configuration or if no configuration has been made yet, click the "Configure" button **(6)**



- Expand the **Advanced Settings** option **(7)** to select the framework version **(8)** for the Communication Channels. This allows backward compatibility for Communication Instances created with previous versions of SMART**UNIFIER**.



- Save the SMART**UNIFIER** Instance by clicking the "Save" button **(9)**
- To deploy, run, and stop the Instance, navigate to the *Deployment* perspective.

CHAPTER

FOUR

CONFIGURATION COMPONENT MANAGEMENT

SMART**UNIFIER** provides a comprehensive management of the configuration components:

- *Group Filter*
- *Validation*
- *Component Version Control*
- *Operations*

In order to keep the SMART**UNIFIER** configuration components organized take a look on *how to name the configuration components*.

Naming Convention

The configuration process of a SMART**UNIFIER** Instance involves the creation of *configuration components* such as Information Models, Communication Channels, Mappings, Device Types and the Communication Instances themselves. Each configuration component needs to have a group and a name as an identifier. The identifier helps to organize configuration components and build up a hierarchical structure.

Group

The group can be used to leverage the concept of paths, similar to those used in file systems, by concatenating logical entities together. In most cases, there is already a naming convention or a certain style of naming for equipment in place. This convention can be adopted when naming and organizing the configuration components.

Name

The name assigned to each configuration component should clearly reflect the entity being integrated. The following table provides examples of appropriate naming conventions for the specific configuration components:

Configuration Component	Description	Naming Example
Information Model	Name of the entity that is modeled	CNC-Machine
Communication Channel	Name of the entity that is connected to	CNC-Machine
Mapping	Name of the source and target entity	CNC-MachineToDatabase
DeviceType	Name of the entity type that is integrated (CNC-Machine)	CNC-Machine
Instance	Specific name of the entity that is integrated (Equipment)	CNC3000

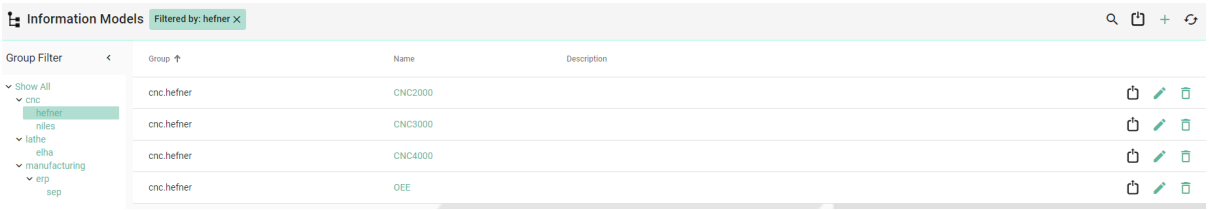
Naming Examples

We recommend the following naming convention for improved comprehensibility.

Information Models & Communication Channels

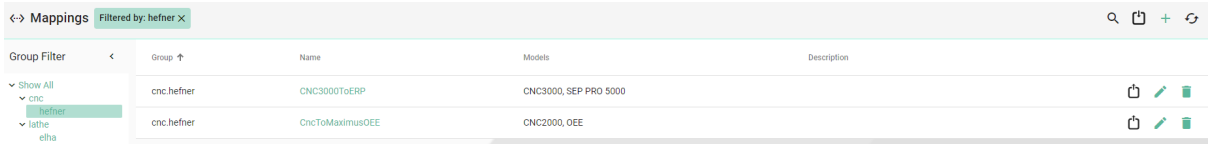
Information Models and its associated Communication Channels should both have the same naming.

System	Group Format	Name Format	Example Group	Example Name
Equipment / Machines	Equipment-Type.Manufactur	Equipment-Name	cnc.hefner	CNC3000
IT-system	System-Type.Manufactur	SystemName	erp.sep	SEP PRO 5000



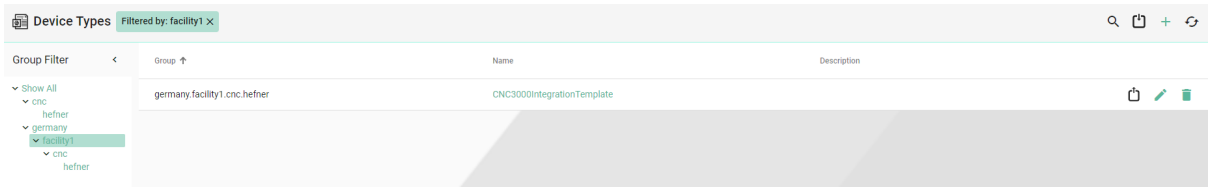
Mappings

System	Group Format	Name Format	Example Group	Example Name
Equipment / IT-system	Equipment-Type.Manufactur	Equipment-ToSystem	cnc.hefner	CNC3000ToERP



Device Types

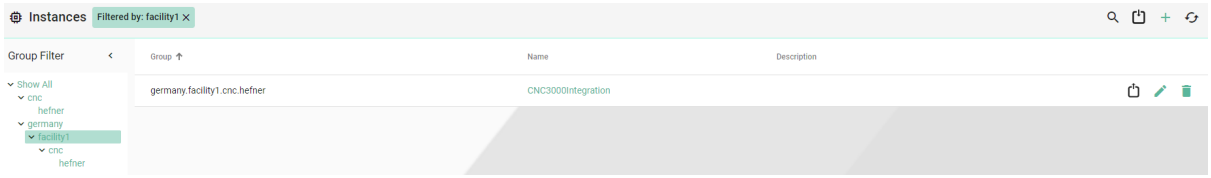
System	Group Format	Name Format	Example Group	Example Name
Equipment / IT-system	Region.PlantName.E	Equipment-ToSystemIntegration	germany.facility1.cnc	CNC3000IntegrationTemplate



Communication Instances

The Communication Instance provides the connectivity between the physical equipment on the shopfloor and IT-systems. It is the most granular configuration component that is created.

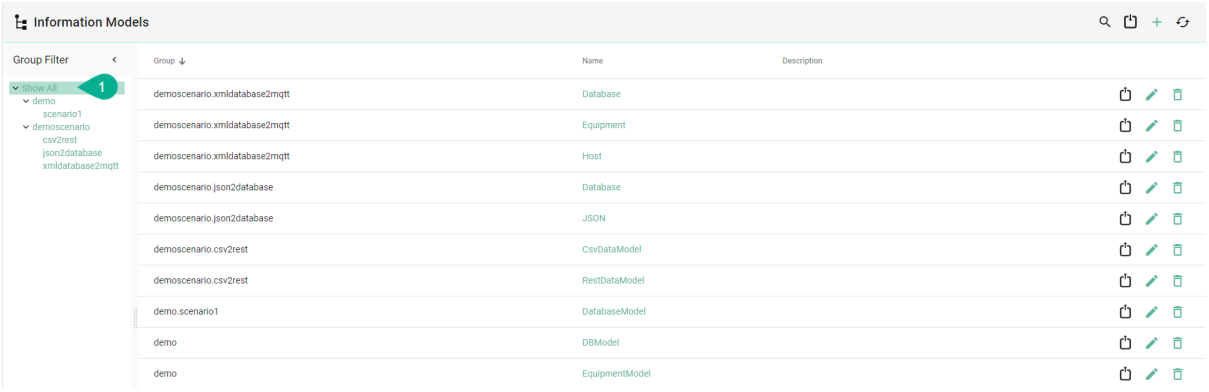
System	Group Format	Name Format	Example Group	Example Name
Equipment / IT-system	Region.PlantName.E	Equipment-ToSystemIntegration	germany.facility1.cnc	CNC3000Integration



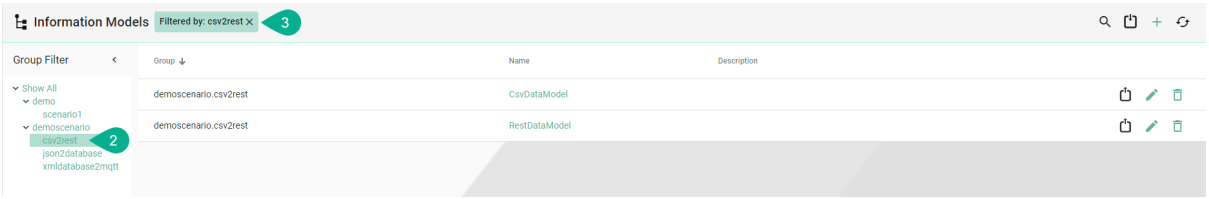
Group Filter

The Group Filter enables easy filtering of configuration components by the group name, which can be hierarchically structured using substrings separated by periods.

The *Show All* filter shows every component **(1)**.



To apply a filter, click one of the items in the Group Filter list **(2)**. The selected filter then becomes visible at the top of the table **(3)**.



The filter can be removed by clicking the selected item again, choosing the *Show All* option, or clicking the cross icon at the filter on top of the table.

Validation

Validation indicates the status of the configuration components such as:

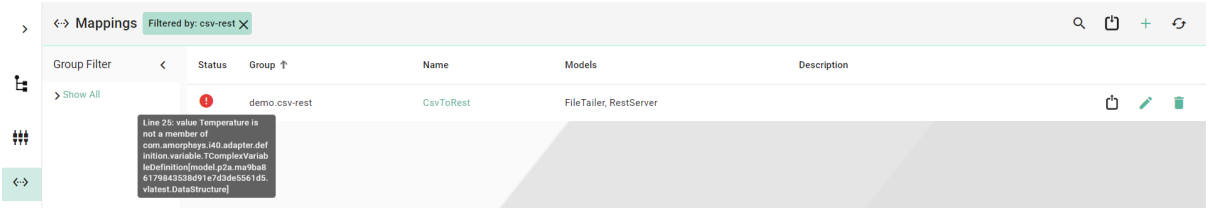
- Information Models
- Communication Channels
- Mappings
- Device Types
- Communication Instances
- Deployment

The status is displayed in the list table of each configuration component.

- If a component is valid, the valid icon is displayed
- If a component is not valid, the error icon is displayed
- If a component is in the process of being validated, the compiling icon is displayed

Example:

If a change is made to, for example, an Information Model, such as altering the data structures or simply renaming nodes, the Mapping where the Information Model is used is recompiled, and the validation status is updated. In this case, an error is displayed with an error message that indicates the problem. The error message is shown when hovering over the error icon.



Component Version Control

Component Version Control enables users to version SMART**UNIFIER** configuration components such as Information Models, Communication Channels, Mappings, Device Types and Communication Instances.

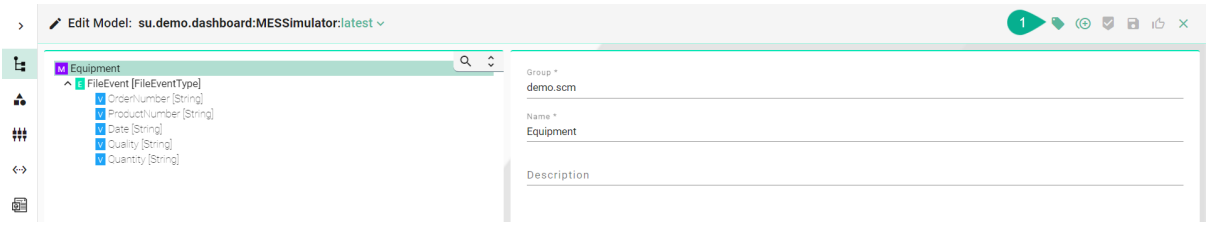
By default, SMART**UNIFIER** is using the Component Version Control internally - therefor no configuration is needed. Another option is to point to an external version control system like Gitea. In order to setup an external version control check out the SMART**UNIFIER** Installation Guide.

How it works: SMART**UNIFIER** creates a repository for each configuration component. Configuration components can be released using tags which reference a specific point in the Git history. After a tag has been created (equivalent to release of a configuration component) there will be no further history of commits/changes. This means that the configuration component can not be edited any further.

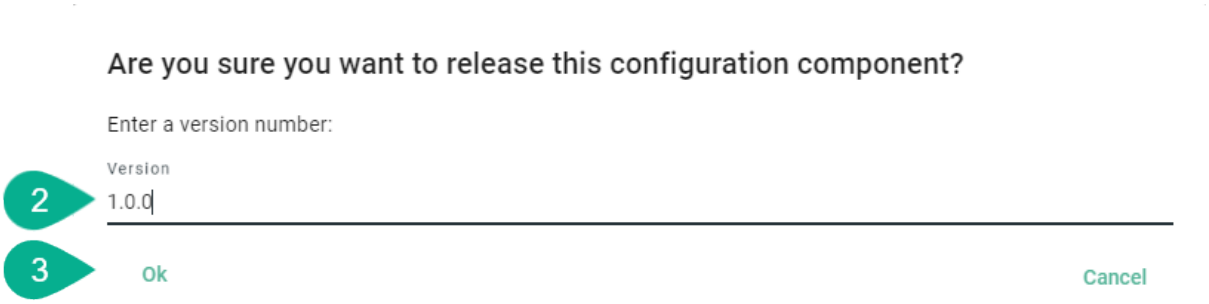
How to release configuration components

In order to release a configuration component follow the steps below:

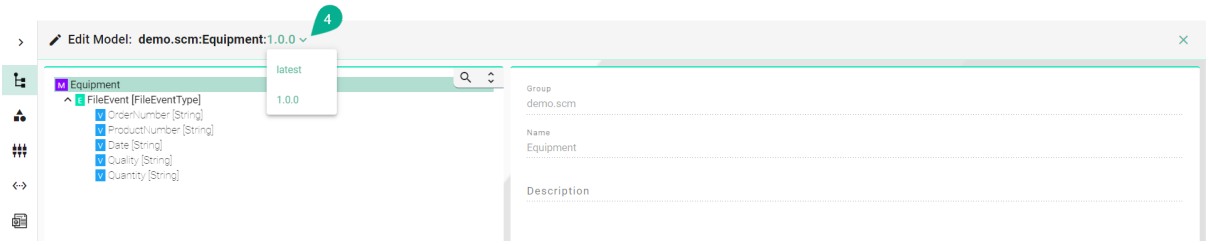
1. Go to an edit page of a configuration component and click the **release** button.



1. Enter a **version number**.
2. Click **Ok** to confirm.



4. Open the version drop-down to change between **latest** and other **tags**.



Note

Once a configuration component is released you can no longer edit the current tag. If changes are necessary select **latest**. Once you finished editing the final version you can repeat the release process as described above.

Operations

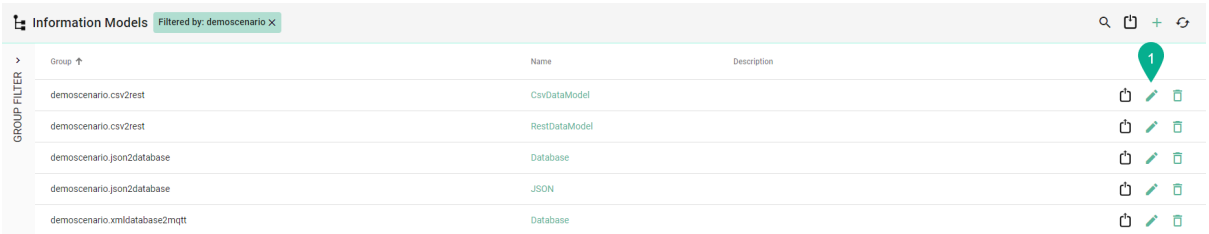
Add

The option to add/create a new component is described in the Instance Setup chapter, for each component type:

- *Information Models*
- *Communication Channels*
- *Mappings*
- *Device Types*
- *Instances*
- *Deployment*
- *Deployment Endpoints*

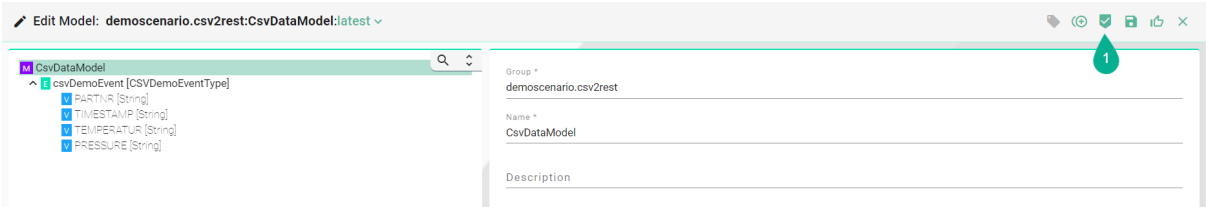
Edit

Components can be edited by clicking the **Edit** button **(1)**.



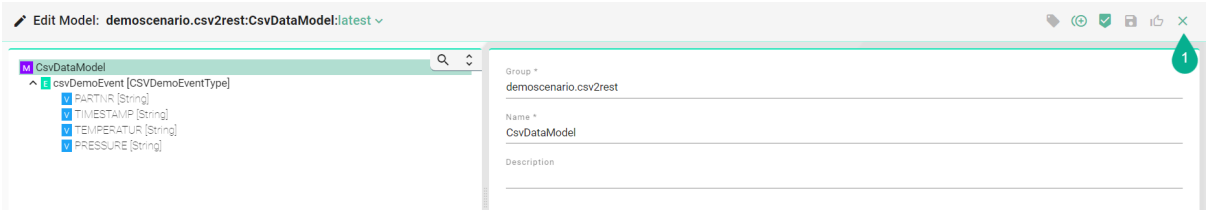
Apply

In edit mode, selecting the **Apply** button **(1)** is required to validate or compile new data input.

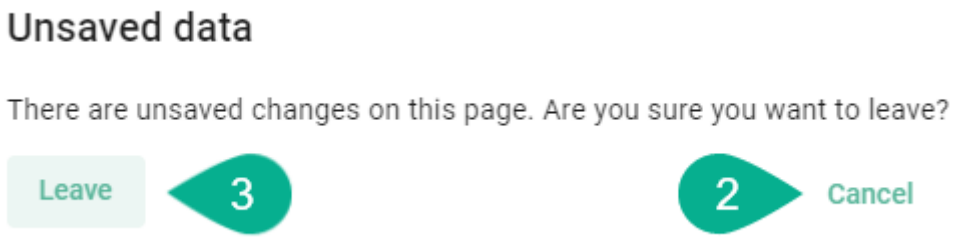


Exit Editing

Edit mode can be exited by clicking the **Close** button **(1)**.

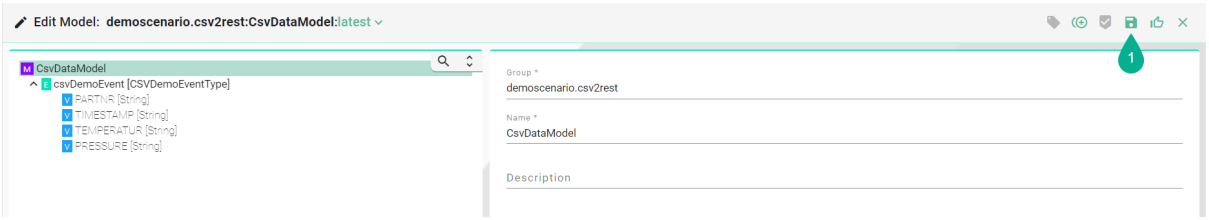


If the data is unsaved, a pop-up will appear, offering the option to click the **Cancel** button **(2)** to return to edit mode and save the data, or the **Leave** button **(3)** to exit without saving.

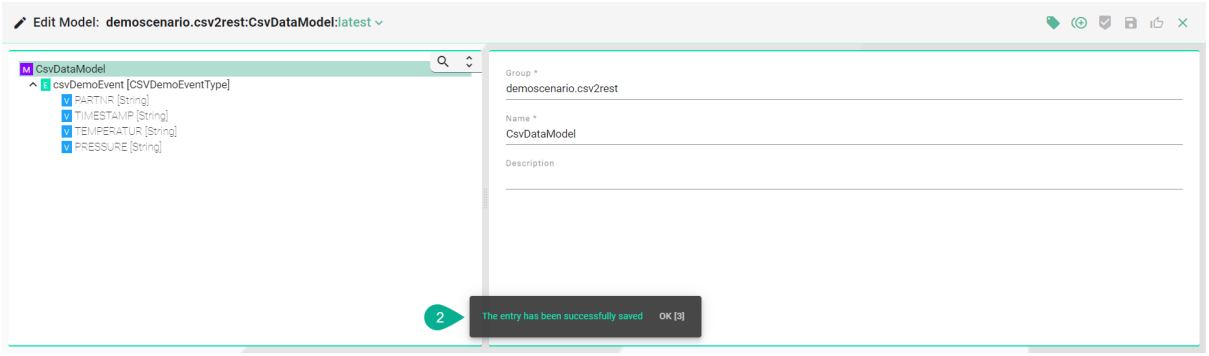


Save

In Edit Mode, changes can be saved by clicking the **Save** button **(1)** after the input data has been applied.

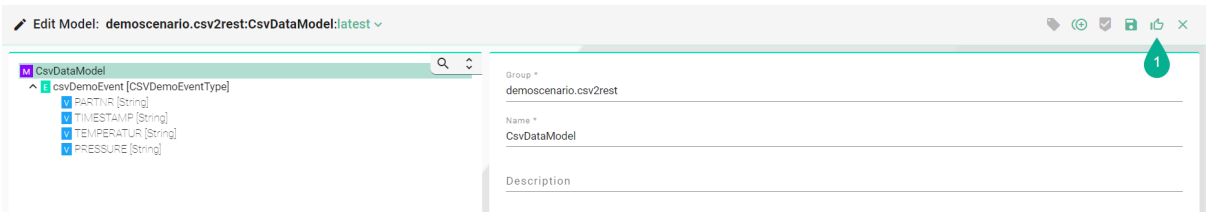


A confirmation message appears **(2)**.

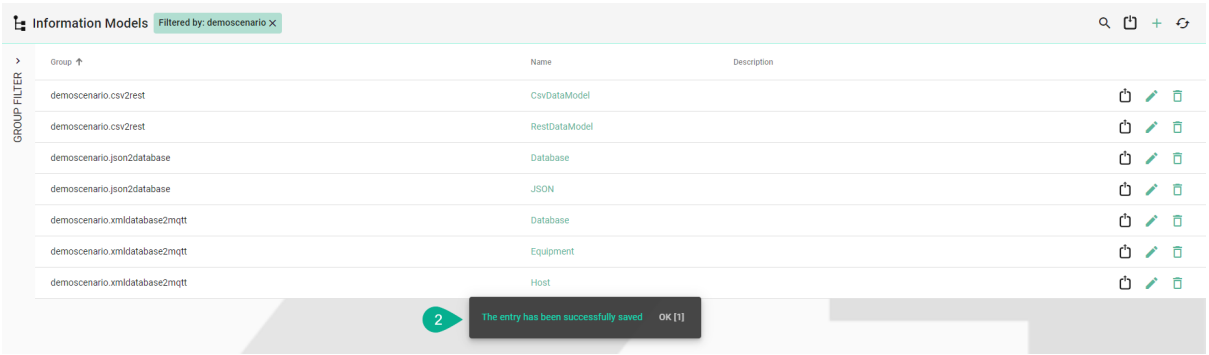


Save and Close

After applying input data while editing a component, changes can be saved and edit mode exited by clicking the **Save and Close** button **(1)**.



A confirmation message appears **(2)**, the edit mode is then closed.



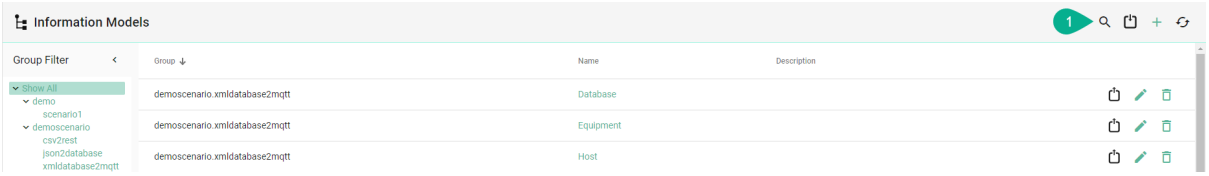
Search

The Search option enables filtering of results according to multiple criteria, including:

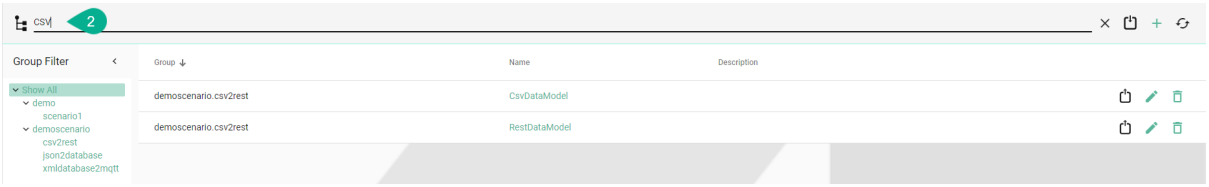
- Name
- Version
- Description

The search is case-insensitive and operates as a partial search, displaying all results that match the searched characters.

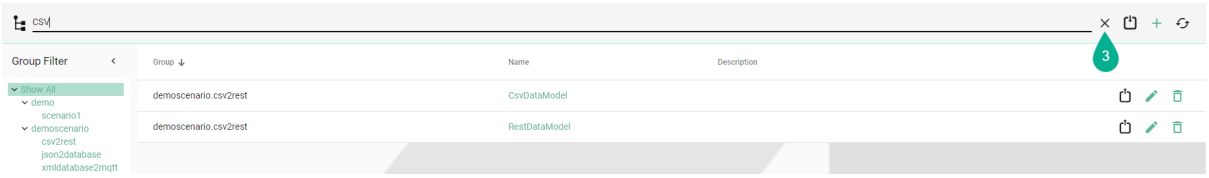
To find a component, click the **Search** button **(1)** located in the upper right corner.



Enter a search term **(2)**.



To cancel the search click on the **Close Search** button **(3)**.

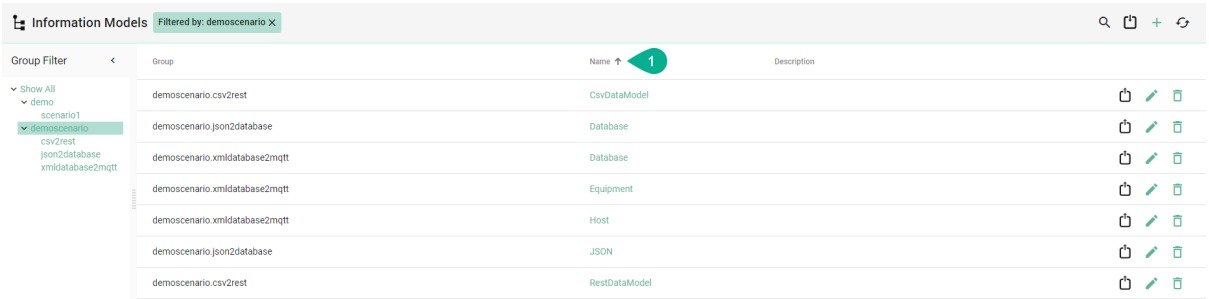


Sort

The information in the view mode can be sorted ascending or descending for each column:

- Group
- Name
- Version
- Description

Click on the column header **(1)** to sort the information within a column. An arrow icon will show whether the components are sorted in ascending or descending order.

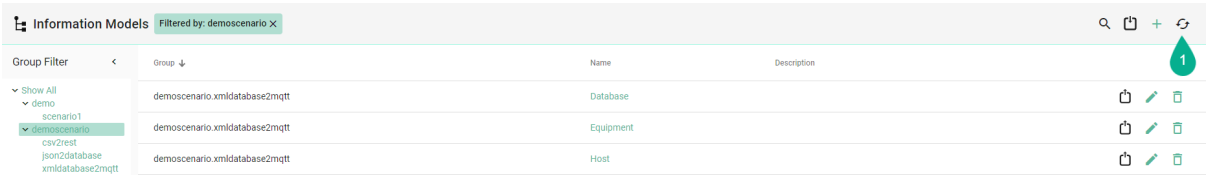


In the table view, next to each component on the right, the following operations are available:

- Export
- Edit
- Delete

Reload

Selecting the **Reload** button **(1)** in the upper right corner refreshes the components from the repository.



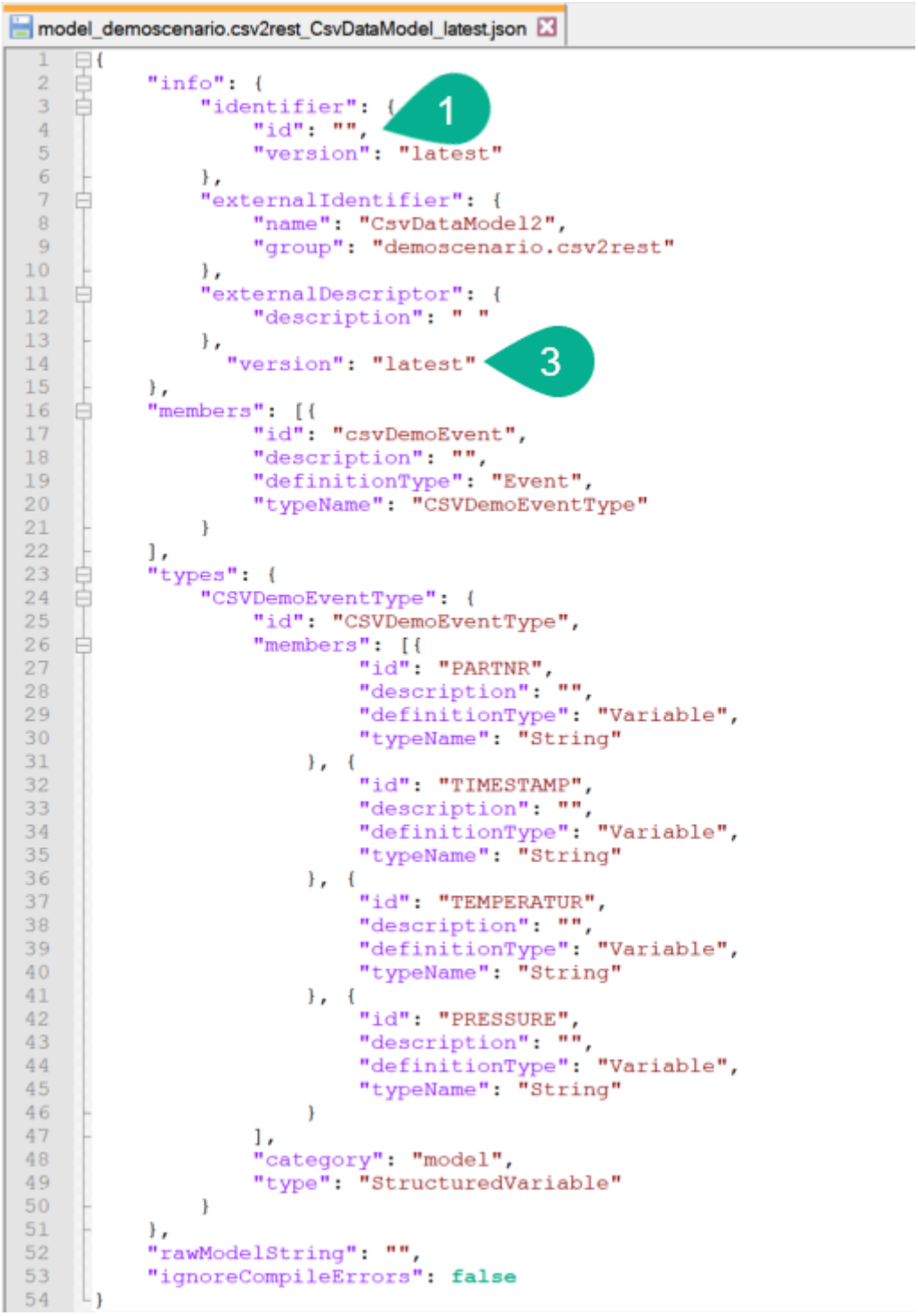
Import

This feature enables the addition of a newly created or exported component to the scenario.

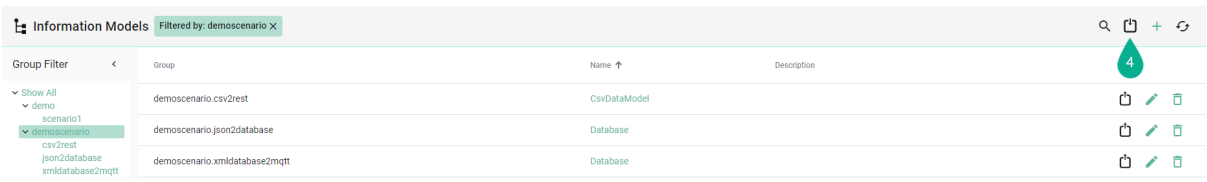
To import an exported component, first open the JSON file and remove the component **id (1)**; upon import, the database will assign a universally unique identifier (uuid). Additionally, copy **(2)** and paste **(3)** the **version** into the **info** section, as illustrated below.

model_demoscenario.csv2rest_CsvDataModel_latest.json

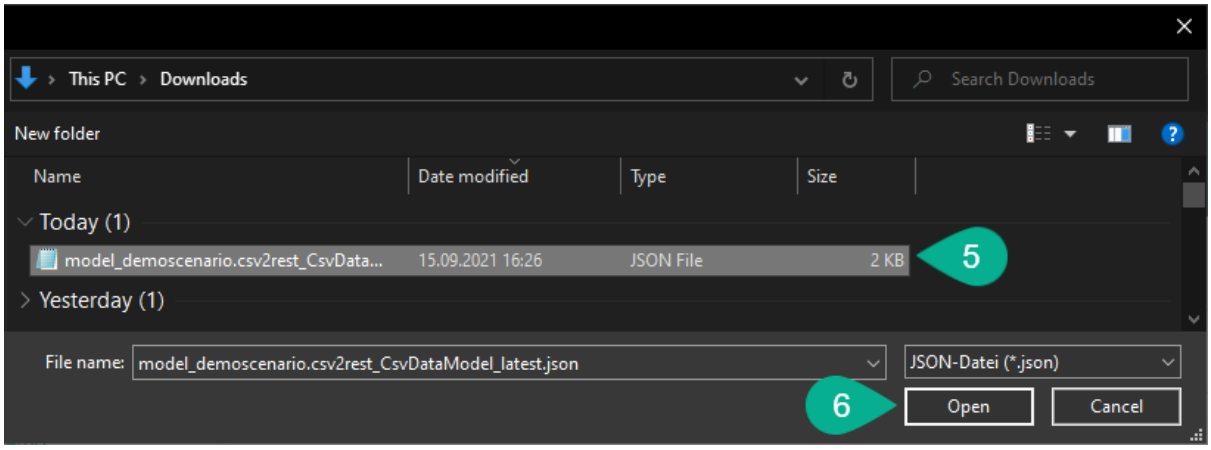
```
{
    "info": {
        "identifier": {
            "id": "8b6bc3f2-192d-4870-8bad-b7c2f5e191b9",
            "version": "latest"
        },
        "externalIdentifier": {
            "name": "CsvDataModel",
            "group": "demoscenario.csv2rest"
        },
        "externalDescriptor": {
            "description": " "
        }
    },
    "members": [{
            "id": "csvDemoEvent",
            "description": "",
            "definitionType": "Event",
            "typeName": "CSVDemoEventType"
        }
    ],
    "types": {
        "CSVDemoEventType": {
            "id": "CSVDemoEventType",
            "members": [{
                    "id": "PARTNR",
                    "description": "",
                    "definitionType": "Variable",
                    "typeName": "String"
                }, {
                    "id": "TIMESTAMP",
                    "description": "",
                    "definitionType": "Variable",
                    "typeName": "String"
                }, {
                    "id": "TEMPERATUR",
                    "description": "",
                    "definitionType": "Variable",
                    "typeName": "String"
                }, {
                    "id": "PRESSURE",
                    "description": "",
                    "definitionType": "Variable",
                    "typeName": "String"
                }
            ],
            "category": "model",
            "type": "StructuredVariable"
        }
    },
    "rawModelString": "",
    "ignoreCompileErrors": false
}
```



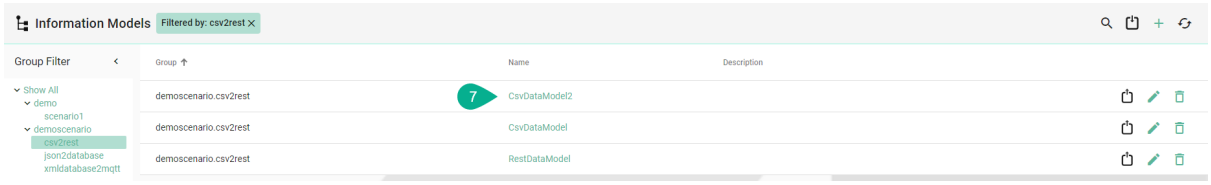
To proceed with the import, click the **Import** button **(4)** located in the upper right corner.



Select the file **(5)** and then click the **Open** button **(6)**.

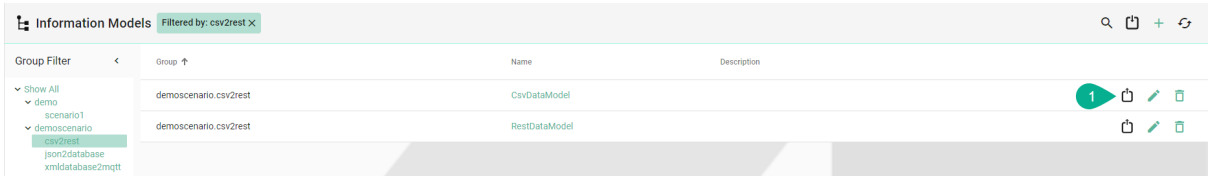


Now, the imported component appears in the list **(7)**.



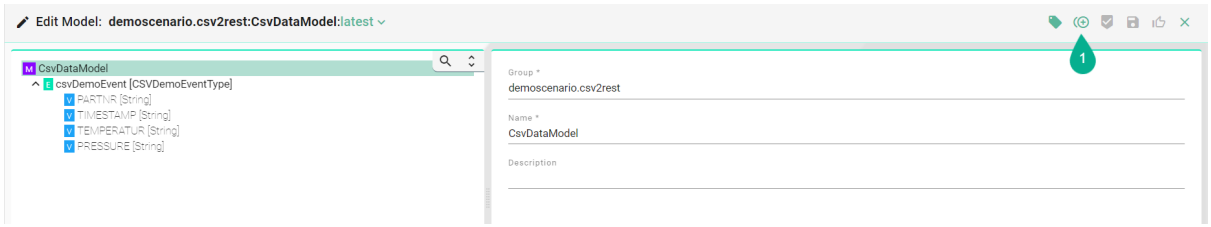
Export

To export a component, click the **Export** button **(1)**.

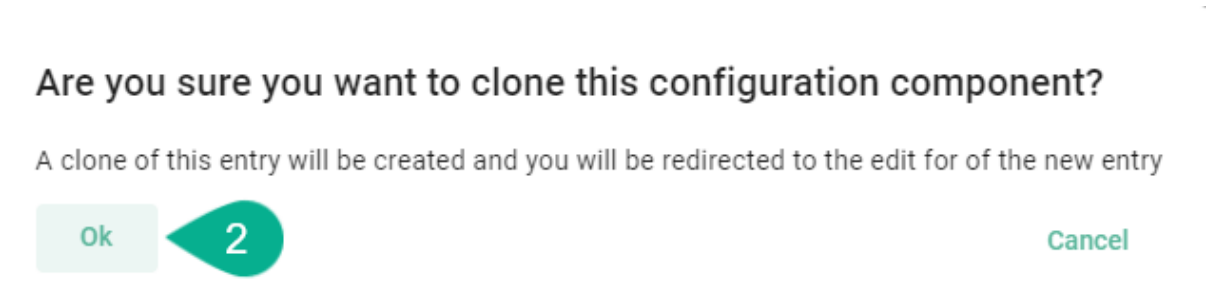


Clone

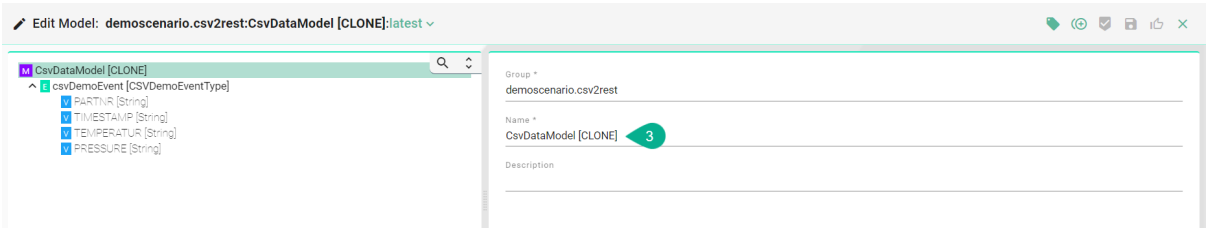
In edit mode, a component can be cloned by selecting the **Clone** button **(1)**.



Click on the **Ok** button **(2)**.



The cloned component is then visible in edit mode, requiring a valid name to be input **(3)**

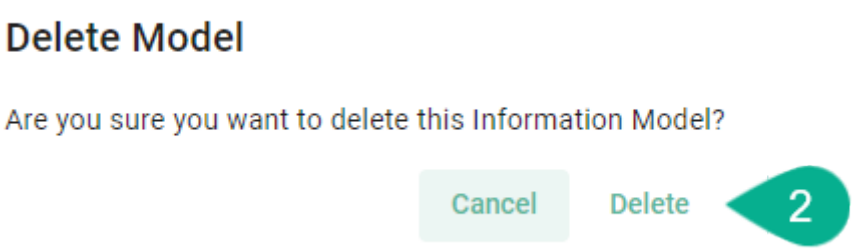


> **Note**
>
> The option to clone is unavailable for the Deployment component.

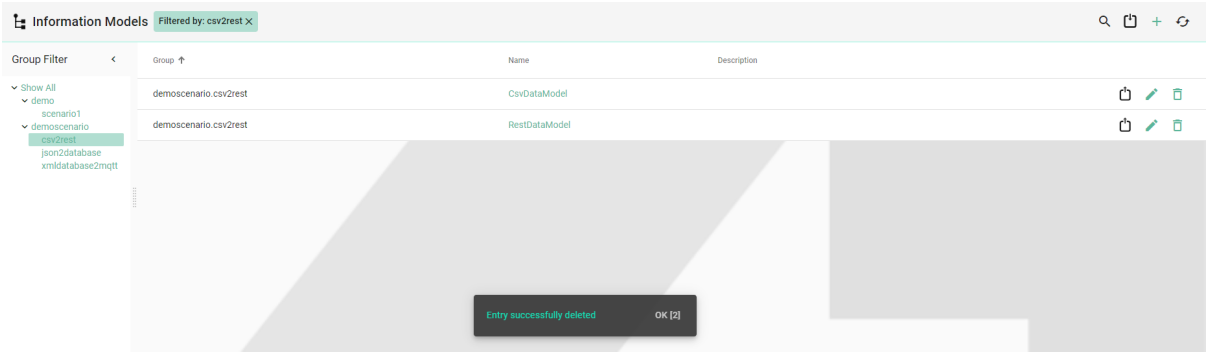
Delete

A component can be deleted by clicking the **Delete** button **(1)**.

Select **Delete (2)**.



The component is deleted and removed from the list.

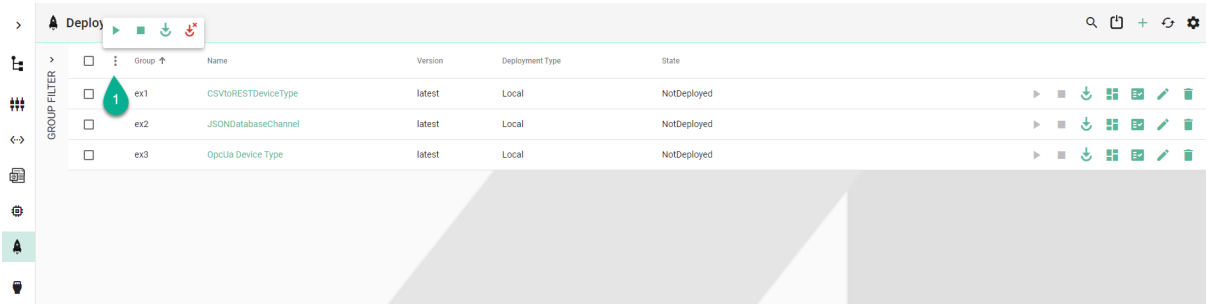


Bulk Action

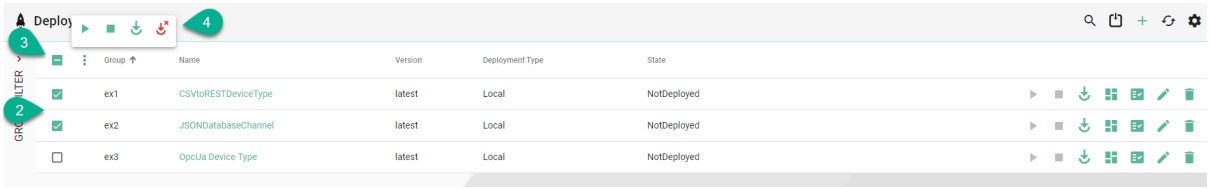
> **Note**
>
> This operation is available only for the **Deployment**.

The following bulk operations are available **(1)**:

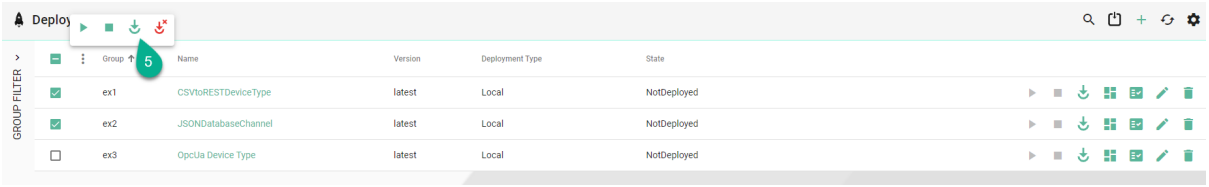
- Start
- Stop
- Deploy
- Undeploy



To begin, tick the boxes for specific Deployment Instances **(2)** or the box to select all **(3)**.Then click on the **ellipsis menu** button **(4)**.

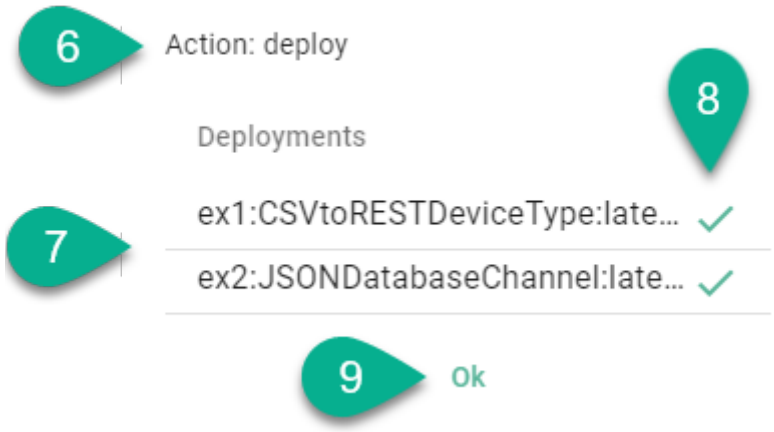


Choose a bulk operation; here, the selected instances should be deployed **(5)**.



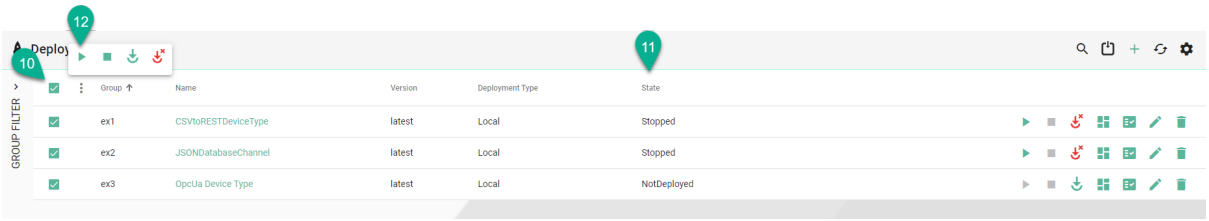
A status popup appears, displaying the following information:

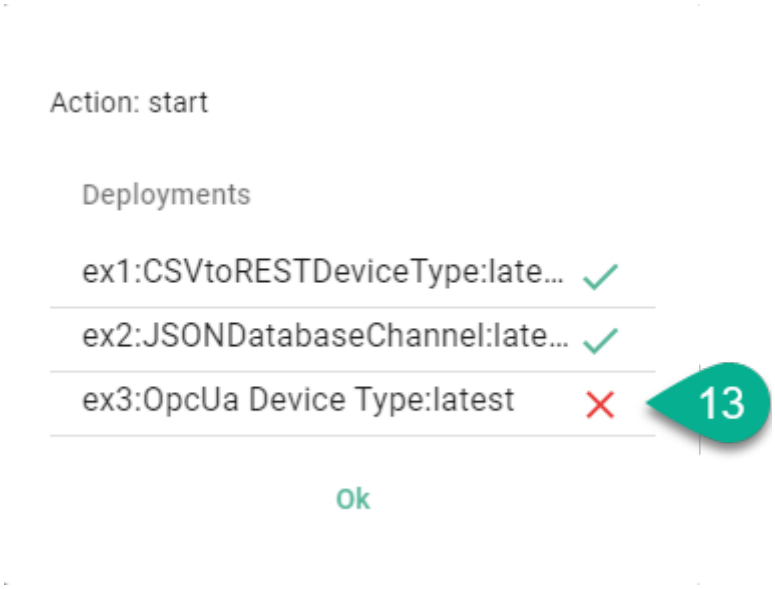
- Performed action **(6)**
- The Instances included in the bulk action **(7)**
- The status of the action **(8)**



Click the **Ok** button **(9)** to close the popup.

When the selected Instances **(10)** are in different states **(11)**, the bulk action **(12)** will only affect Instances with the compatible states **(13)**.





Note

Instances which are *protected* will not work using bulk actions.

CHAPTER

FIVE

DEPLOYMENT

SMART**UNIFIER** supports the *deployment* of Instances on several computing environments:

- *Local* - on the same environment the SMART**UNIFIER** Manager is running on
- *Agent Process* - remote on any machine

Learn how to *operate* and *monitor* your SMART**UNIFIER** Instances.

Learn about *notifications*.

Learn about additional *deployment options*.

Deployment Types

You can deploy your SMART**UNIFIER** *Communication Instances* to any IT resource (e.g., Equipment PC, Server, VM, Cloud) suitable to execute SMART**UNIFIER** Instances. Please refer to the installation guide for the system requirements. To see how an Instance is deployed, choose one of the **deployment types** below:

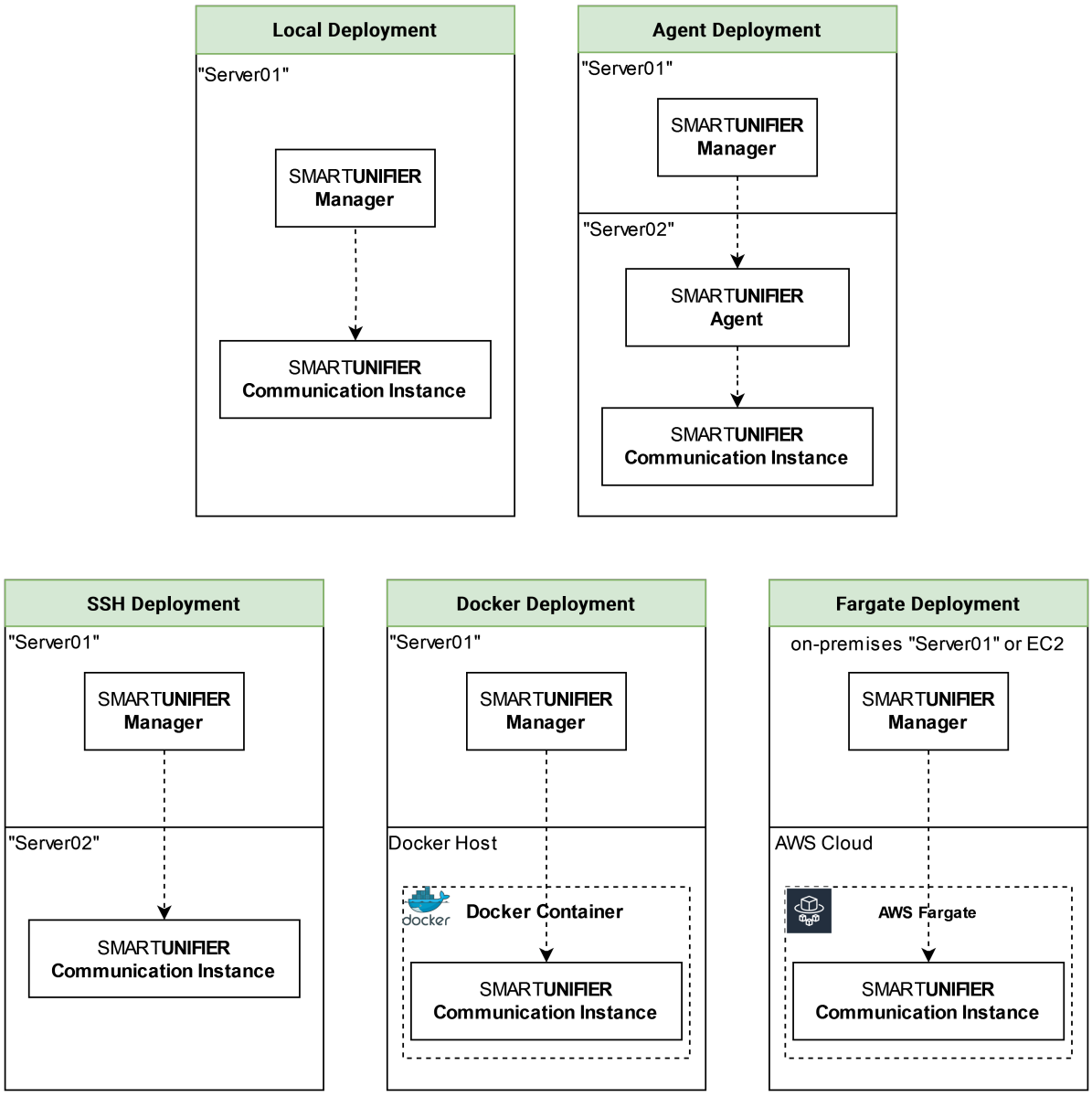


Fig. 1: Overview over Deployment Types of SMART**UNIFIER** Communication Instances

Local and Agent Deployment

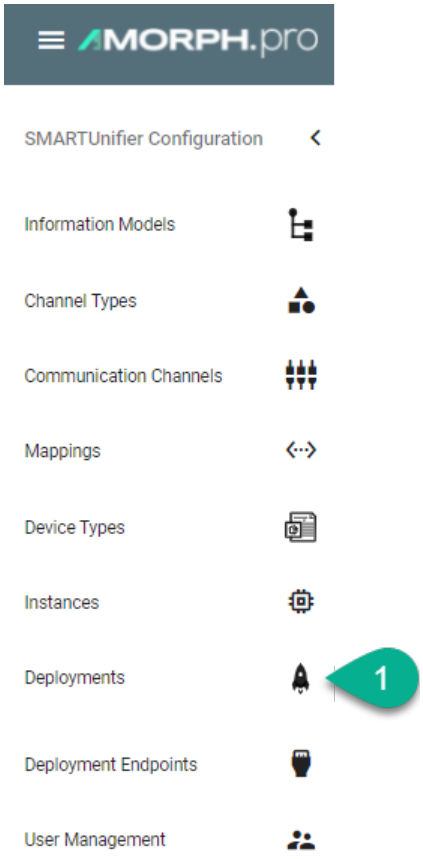
SMART**UNIFIER** Communication Instances can be deployed on the IT-resource where the SMART**UNIFIER** Manager is running on (e.g., a computer, a server or the AWS Cloud).

Hint

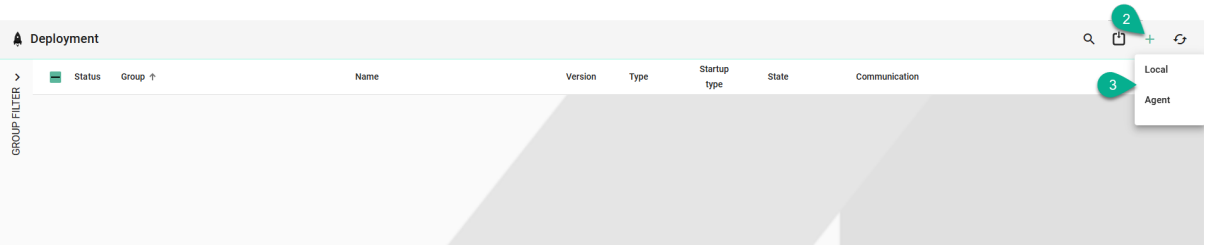
Before deploying a Communication Instance make sure to create and **Start** a *Deployment Endpoint*. The Deployment Endpoint specifies the location where you want the Instance to run. By default, the SMART**UNIFIER** package already comes with a Local Deployment Endpoint preconfigured.

Follow the steps described below in order to deploy a local/ agent Communication Instance:

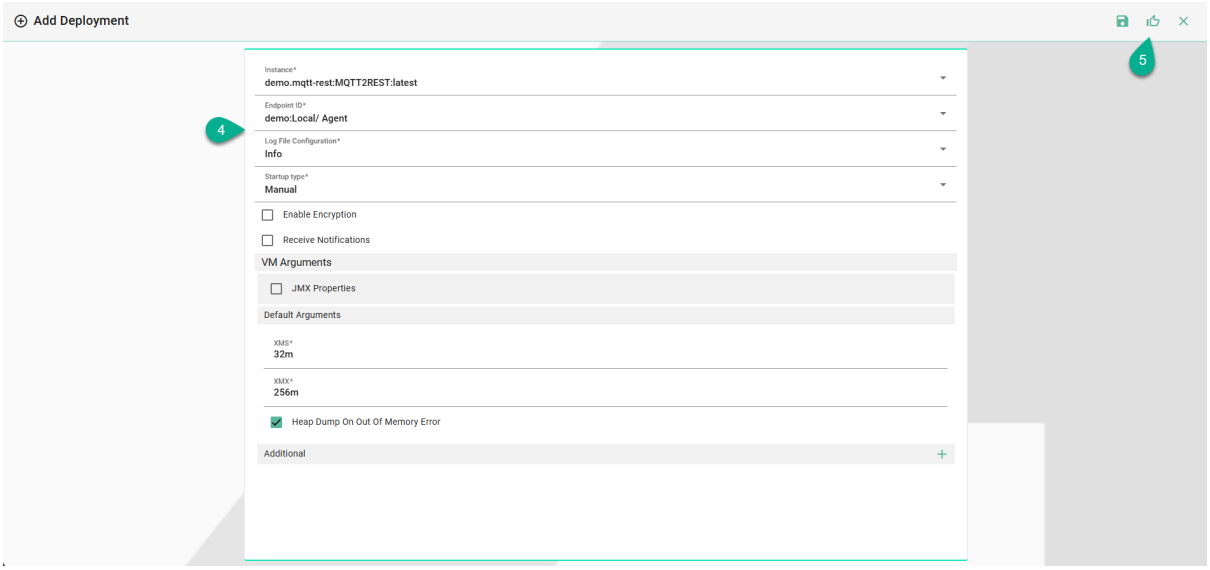
- Select the SMART**UNIFIER** Deployment perspective **(1)**.



- Click on the **Add Deployment** button **(2)**.
- Select the Deployment Type **Local** or **Agent** from the pop-up **(3)**.



- In the Add Deployment view a set of configuration parameters is required **(4)**
  - Select the SMART**UNIFIER** Communication Instance to be used in the Deployment.
  - Either select the **Local Endpoint**, or select the agent Endpoint ID created in the *Agent section* from the Drop-Down menu.
  - Select the *log file level*. We recommend the log level of type *Info* in case of a normal deployment scenario.
  - Select the **Startup Type** [Manual|Automatic|Disabled], Manual is set as default.
  - (Optional) Enable *Encryption*.
  - (Optional) Receive *Notifications*.
  - (Optional) Add *VM Arguments*.
- When all mandatory fields are filled click the **Save and Close** button **(5)**.



When the Instance is deployed locally, it's configuration will be copied in the **Deployment folder** defined in the *Local Deployment Endpoint configuration*.

Note

The Instance configuration folder can be copied to another location and started, but the Instance will not be monitored by the SMART**UNIFIER** Manager.

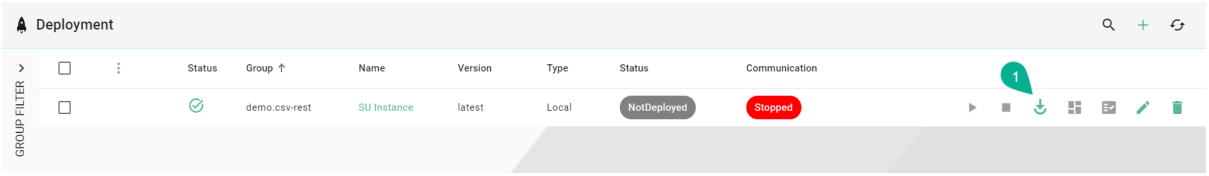
Deployment Endpoints

- *Local*: Deployment of a Communication Instance to your local computer where the SMART**UNIFIER** Manager is running on.
- *Agent*: Deployment of a Communication Instance remote on any machine.

How to Deploy, Run and Operate a Deployed Instance

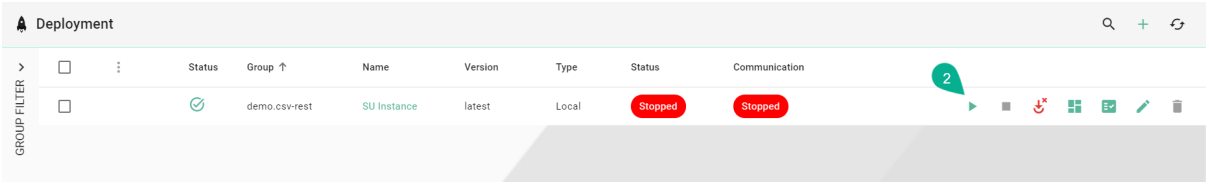
How to Deploy an Instance

- In order to start the Instance, click first the "Deploy" button **(1)**. A message is shown, that confirms the successful deployment of the Instance.



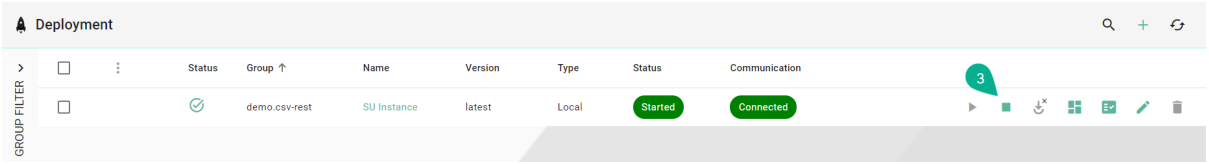
How to Run an Instance

- After successfully deploying the Instance, the state changes from *NotDeployed* to *Stopped*. You can now click the enabled "Start" button **(2)**. The Instance state will change to *Started*. A message is shown, that confirms the successful start of the Instance.
- Also the Communication state changes from *Stopped* to *Started*. If the Communication state remains *Stopped*, please check the log files for errors.



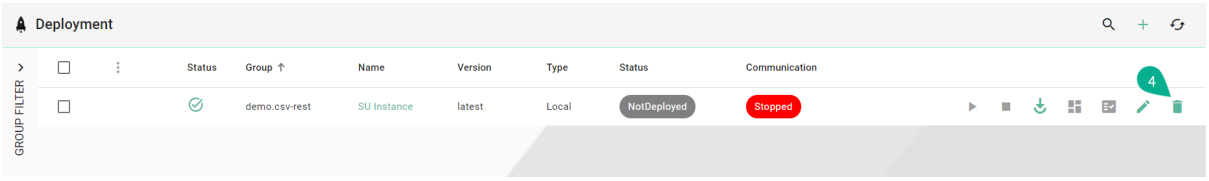
How to Stop an Instance

- To stop the Instance, click the "Stop" button **(3)**.



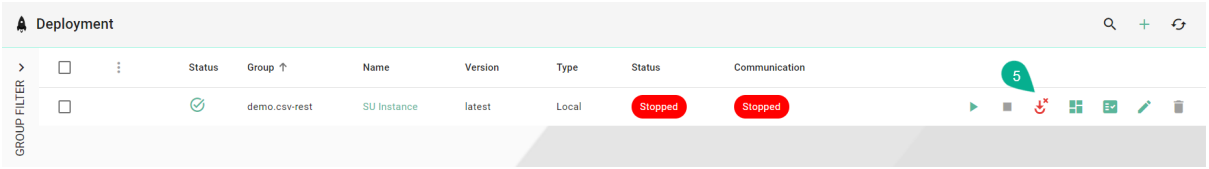
How to Delete a Deployment of an Instance

- Click on the "Delete" button to delete the Deployment **(4)**. This is only possible if the Instance state is *NotDeployed*.

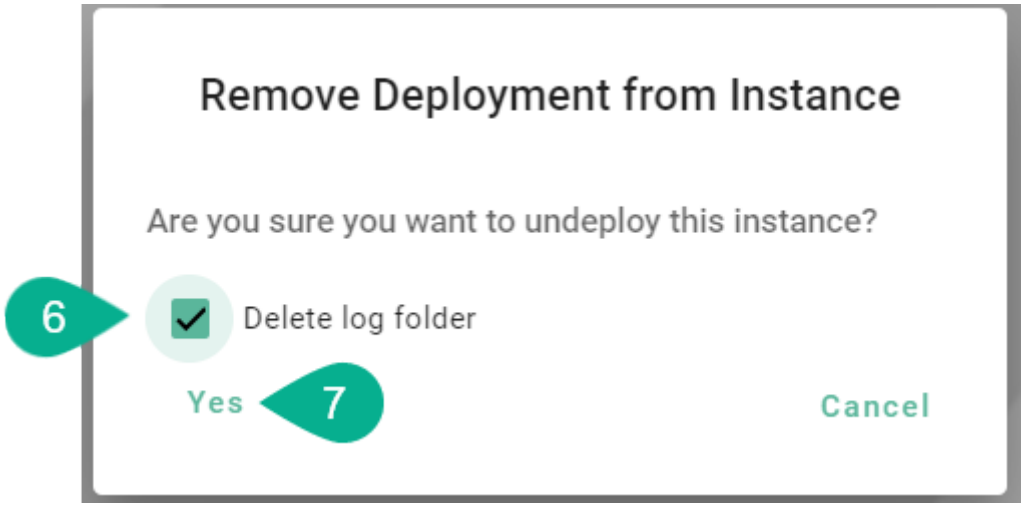


How to Un-deploy an Instance

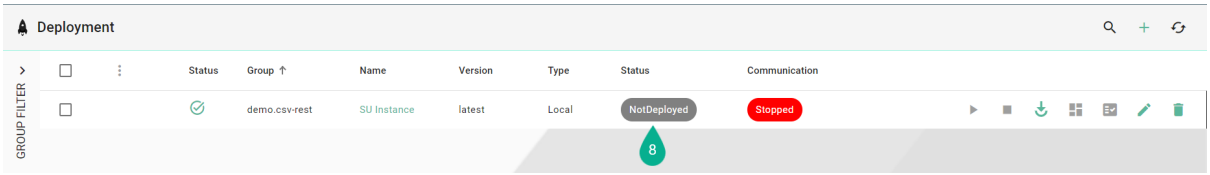
- To undeploy an instance, ensure that it is not running. If necessary *stop the Instance*.
- Click the "Undeploy" button **(5)**.



- A popup appears. Uncheck box **(6)** to keep the log folder
- Then click the **Yes** button **(7)** to confirm.

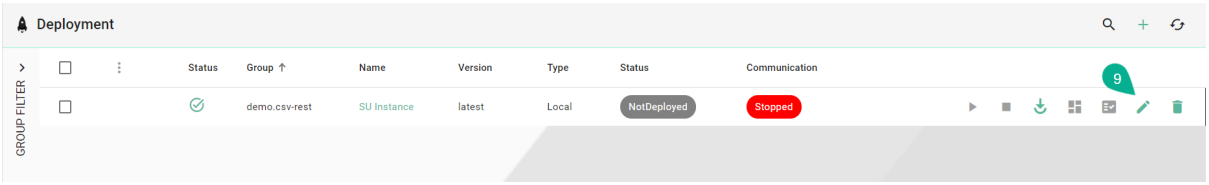


- The Instance state changes to *NotDeployed* **(8)** and the Deployment can be edited.
- If the Instance is in the *NotDeployed* state, changes can be made to the used *Communication Channels* in the *Instance configuration*. Be sure to deploy the Instance again after making changes.



How to Edit a Deployment of an Instance

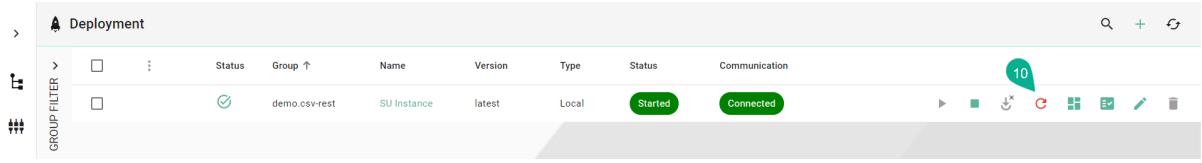
- Click on the "Edit" button to perform changes to the Deployment **(9)**. It is only possible to edit a Deployment if the Instance is not deployed. In case the Instance is deployed, only the details of the Deployment can be viewed.



How to Redeploy a Deployment of an Instance

- Enable **Debug** mode with **F9**
- Click on the "Edit" button to perform changes to the **Redeploy** button **(10)**

> **Hint**
>
> When making changes to an instance's configuration, this method provides an easy way to quickly redeploy the instance with the new settings.



How to monitor Communication Instances

> **Note**
>
> Monitoring a Communication Instance is only possible after it has been deployed.

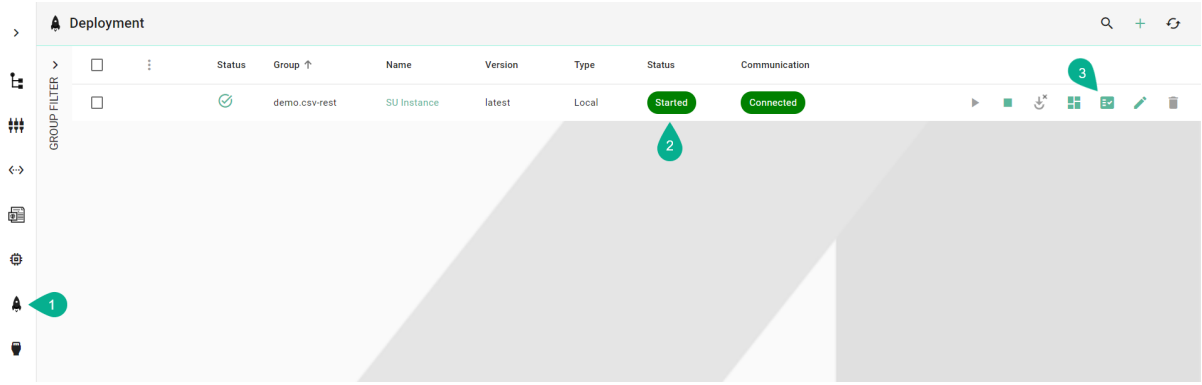
Log Viewer

SMART**UNIFIER** includes an integrated Log Viewer, offering insights into deployed and operational Communication Instances.

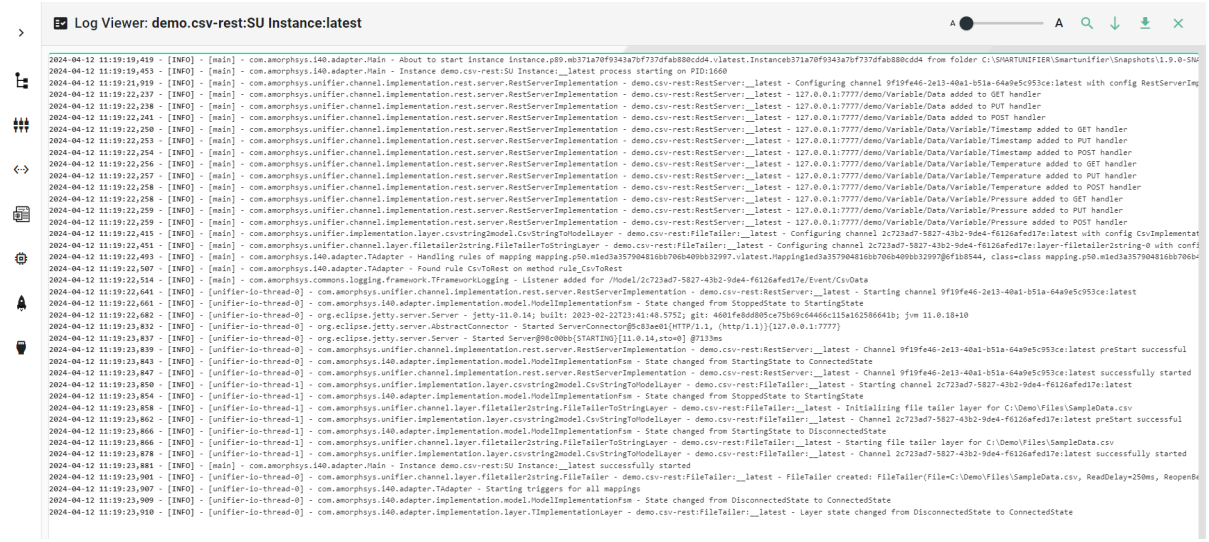
Accessing the Log Viewer

To access the Log Viewer, follow these steps:

- Navigate to the Deployment view **(1)**
- Ensure the Instance is deployed and started **(2)**
- Click on the **Log** button **(3)**



- The Log Viewer displays logs for a deployed Communication Instance.



Log Levels

The Log Viewer displays log details based on the level defined during the creation of the deployment:

- **TRACE** - Provides the most detailed information, used only in rare cases for full visibility into the operations of a Communication Instance
- **DEBUG** - Offers less detail than TRACE but more than what is typically needed in a production environment. The DEBUG log level is suitable for troubleshooting issues with a Communication Instance or for use in a test environment
- **INFO** - The standard log level for regular deployment of a Communication Instance, providing essential operational information
- **WARNING** - This log level signifies an unexpected occurrence within a Communication Instance that may lead to issues in communication

Structure of a Log Entry

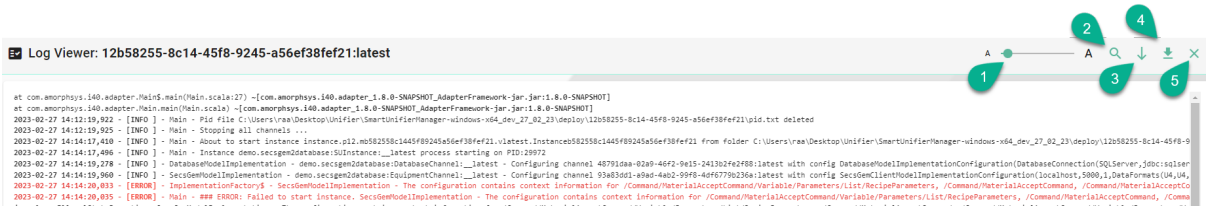
The following table shows the structure of a log entry for a communication instance using two example log entries.

Timestamp	Log Level	Thread	Class	Description
2024-02-19 16:53:16,663	[INFO]	[main]	com.amorphsys.i	Instance demo.csv-rest:SU Instance:__latest successfully started
2024-02-19 16:53:16,713	[ERROR]	[unifier-io-thread-0]	com.amorphsys.u	demo.csv-rest:File:__latest - File C:DemoFilesSampleData.csv does not exist

Log Viewer Features

The Log Viewer offers several features for enhanced usability:

- Adjust Font - Use the slider to change the size of the log font **(1)**
- Filter - Apply a search using a regular expression (Regex) to find specific log entries **(2)**
- Scroll to End - Engage the Follow Tail feature to automatically display the most recent log lines **(3)**
- Download Logs - Download the logs as a ZIP file onto your computer for further analysis or sharing **(4)**
- Close - Exit the Log Viewer and go back to the Deployment perspective **(5)**



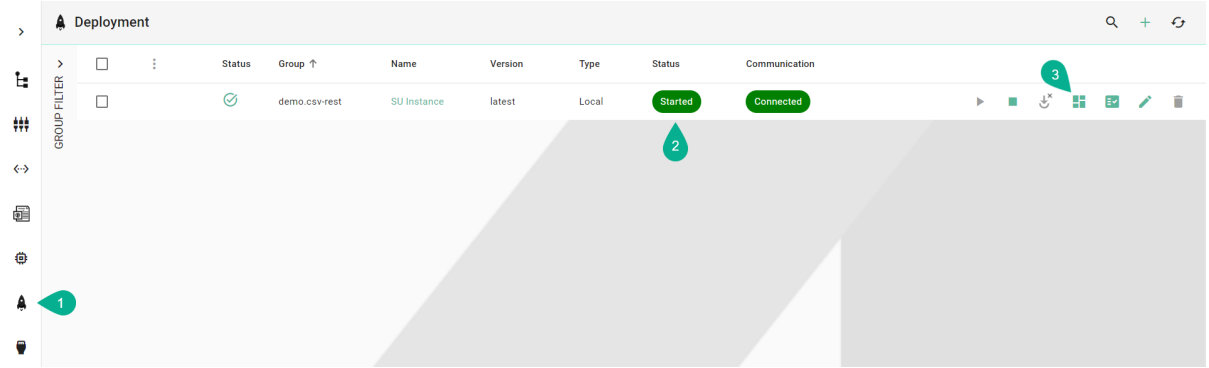
Dashboard

SMART**UNIFIER** provides a Dashboard with an integrated Log Viewer, which helps to gain insights in running Communication Instance performance.

How to access the Dashboard

Follow the steps bellow to access the Dashboard:

- Select the SMART**UNIFIER** Deployment perspective **(1)**
- Make sure the Instance is Deployed **(2)**
- Click on the **Dashboard** button **(3)**



Dashboard's Data

The Dashboard presents a overview of an Instance, showcasing the following key information:

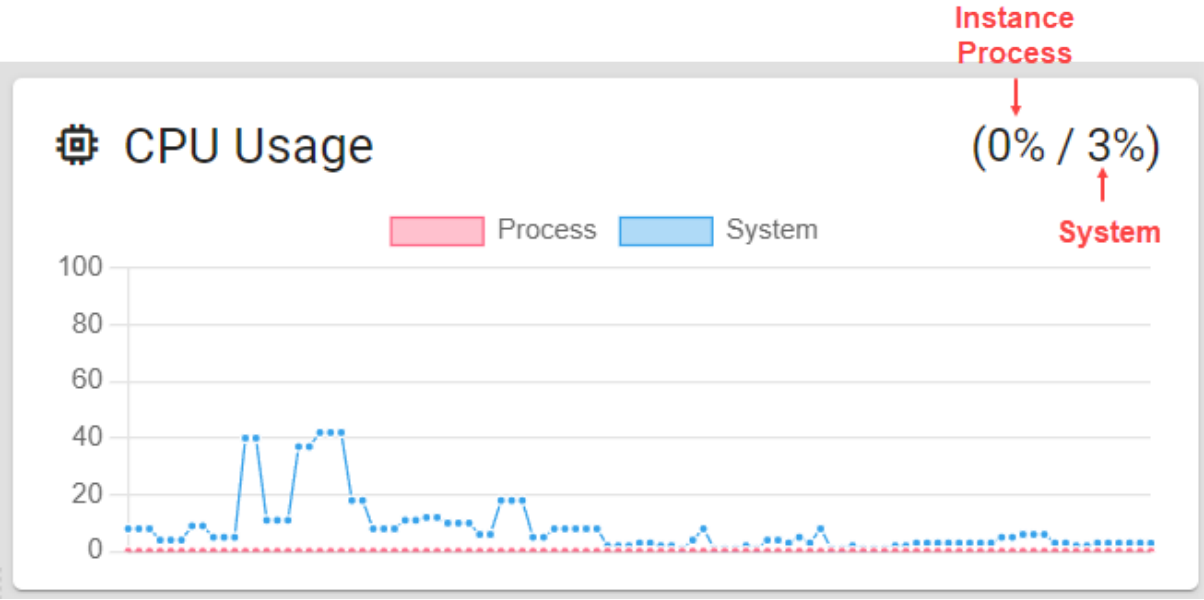
1. Information about *Channels* that are used in the Instance:
   - **Info** - Name of the Communication Channel
   - **Type** - Communication Channel *Type*

- **Status** - Current connection status of the Channel
- **Model** - Associated *Information Model*
- **Messages** - Number of messages *last second* / Number of messages *last hour* (e.g. 30/500)

Hint

The quantity of messages is determined by the Node Types used in the Information Model. Each occurrence of an Event or Command is considered as a single message. On the other hand, when it comes to variables, each individual variable is counted as a message, given that it has been configured within the designated communication channel.

2. Integrated *Log Viewer*
3. Status of the Instance:
   - **Status** - Shows if the Instance is currently *started* or *stopped*
   - **Communication** - Shows the status of the Communication Channels *Connected* or *Stopped*
   - **Start time** - Shows the time when the Instance was started
   - **Time Up** - Shows the duration or uptime of the instance
4. **CPU Usage** - Instance CPU usage in % and system CPU usage in %

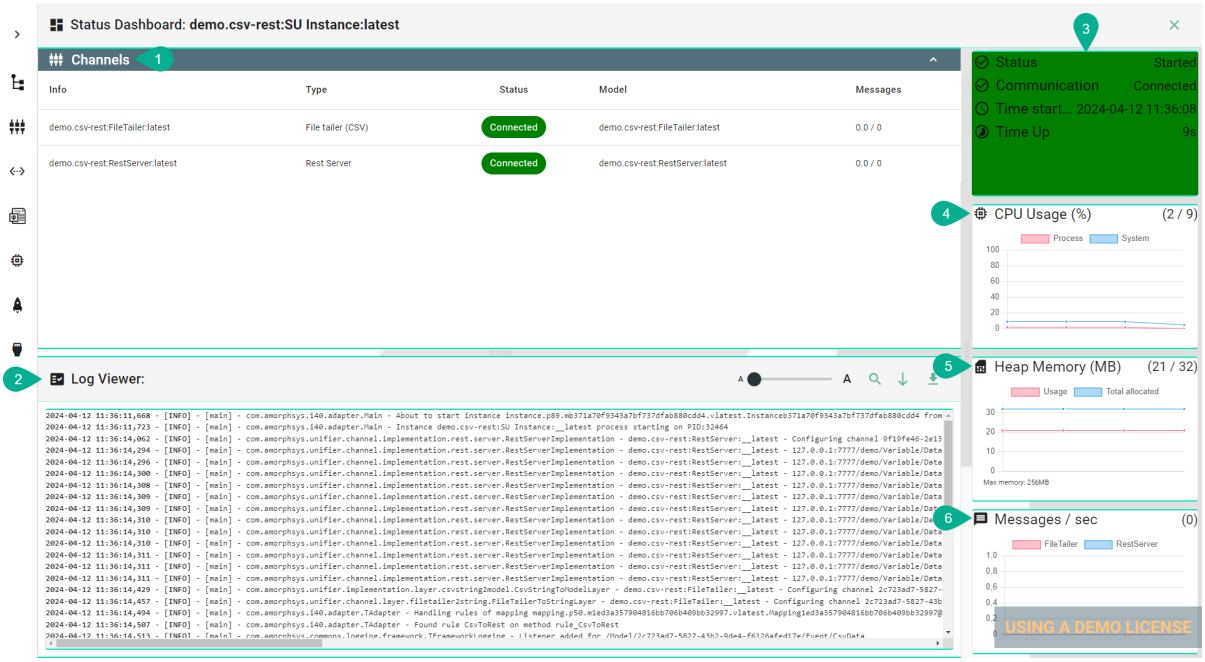


5. **Memory Usage** - Instance Java memory heap usage in MB (megabyte)

Hint

Constant increase in the Java heap space memory might indicate memory leaks that requires

6. **Messages / sec** - Number of total messages of all Communication Channels within the last second



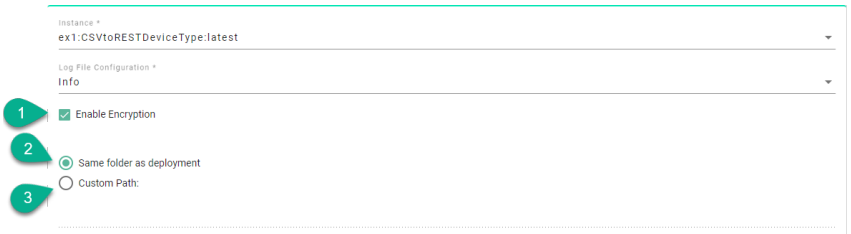
Additional Options

Encryption of Communication Instances

This feature provides the possibility to encrypt the configuration files of Communication Channels used by the Instance, which may contain credentials to access a database or external services. The encryption method used is Advanced Encryption Standard (AES).

The encryption is available for all deployment options, by following the steps bellow:

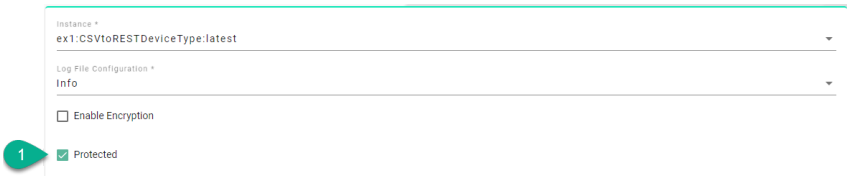
- Check the **Enable Encryption** box **(1)**.
- A symmetrical key (cfg.key) is generated and can be saved in the same folder as the deployment **(2)** or check the **Custom Path** option **(3)** to save the key into a secured location.



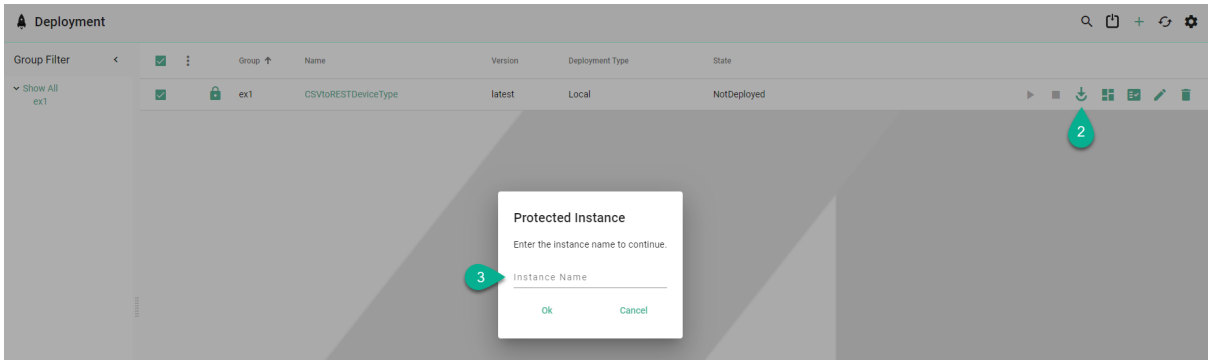
Protect Communication Instances

This feature provides an additional protection when performing an Instance action (e.g., deploy, undeploy, start, stop).

The protection is available for all deployment options, by checking the **Protected** box **(1)**.



Now the Instance is protected, meaning that when the user performs an action like Deploy **(2)**, a popup appears requiring to input the Instance name **(3)**.



Note

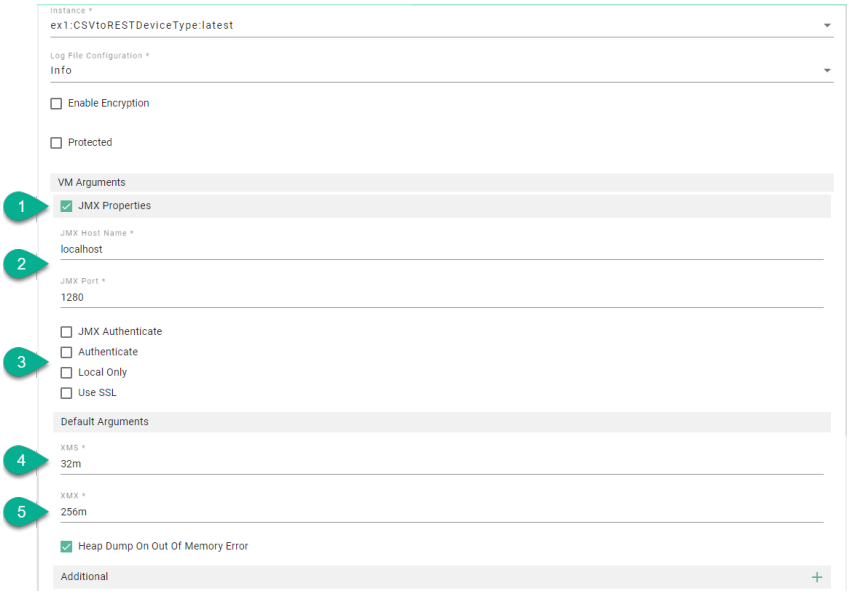
Protected Instances will not work with Bulk actions.

VM Arguments

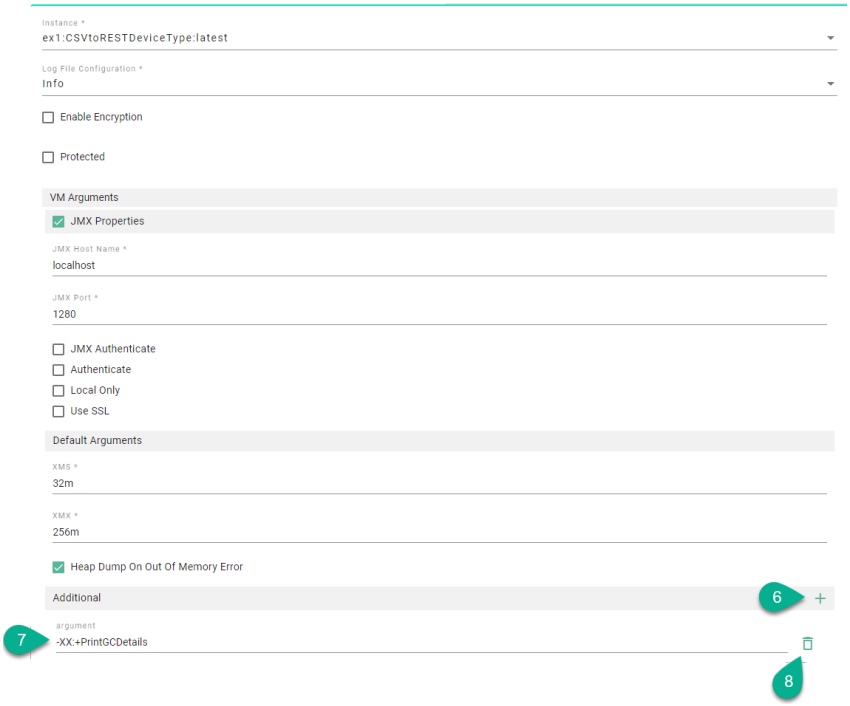
This feature provides the possibility to configure the Java Virtual Machine (JVM). In some cases, when dealing with larger files when using the File Reader Communication Channel (large XML file), it might be necessary to increase the **XMX** in order to avoid running into a *java.lang.OutOfMemoryError* - exception.

VM Arguments can be configured when deploying an Communication Instance locally or on Docker, by following the steps below:

- Check the **JMX Properties** box **(1)** to expand the Java Management Extensions parameters and input the **JMX Host Name** and **Port (2)**.
- Check the authentication method **(3)**.
- Update the **XMS** value **(4)**, minimal heap size, representing the amount of memory used by the JVM to start with.
- Update the **XMX** value **(5)**, maximal heap size, representing the maximum amount of memory that JVM will be able to use.



- By default, the **Heap Dump On Out Of Memory Error** option is checked, providing an analysis file for debugging.
- Additional JVM arguments can be added by selecting the **add Arg** button **(6)** and input the argument **(7)**. For example, to debug memory issues or application performance, the Garbage Collection logging can be enabled in JVM, as seen below.



- An additional argument can be deleted by clicking on the **delete Arg** button **(8)**.

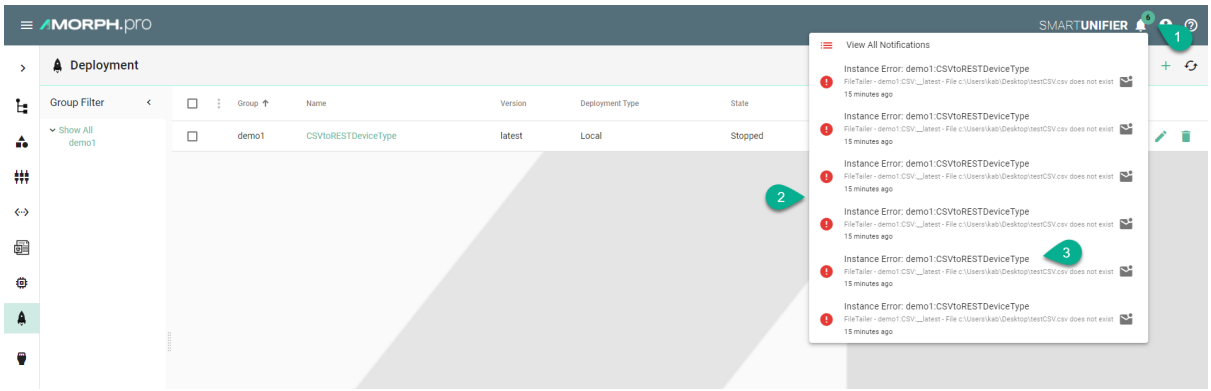
Notifications

SMART**UNIFIER** comes with an integrated notification system, which helps to gain insights when a deployed Communication Instance is started or running and errors appear.

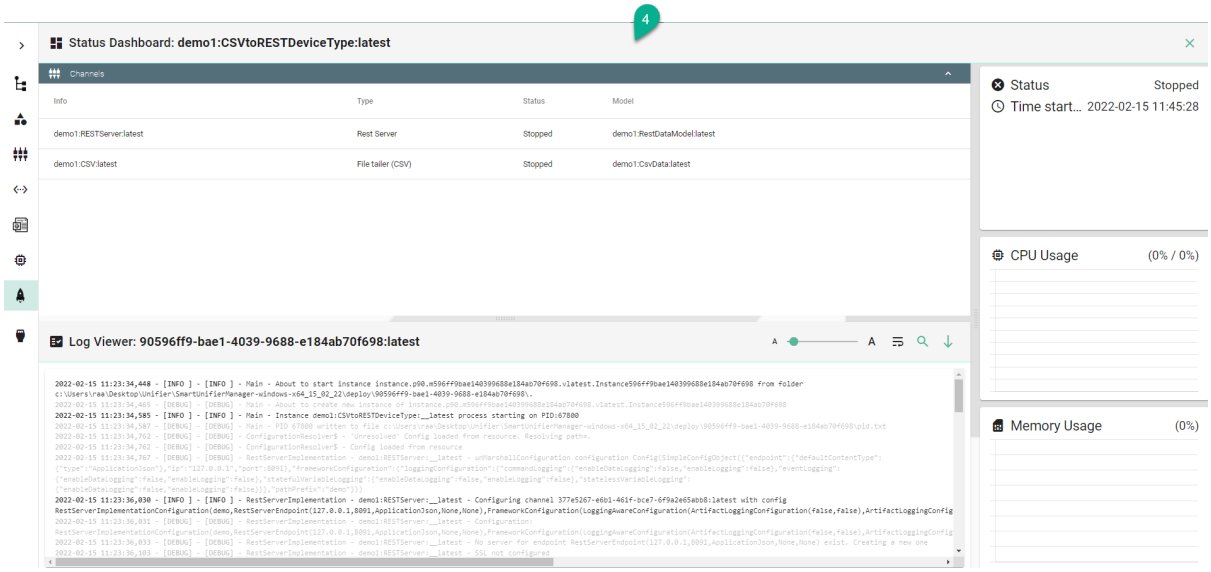
How to access Notifications

When a deployed Communication Instance is started or running and errors appear, the number of errors will be displayed near the **Notifications** button **(1)**.

Click on the Notifications button and the **Notifications List (2)** will display all the Instance errors.

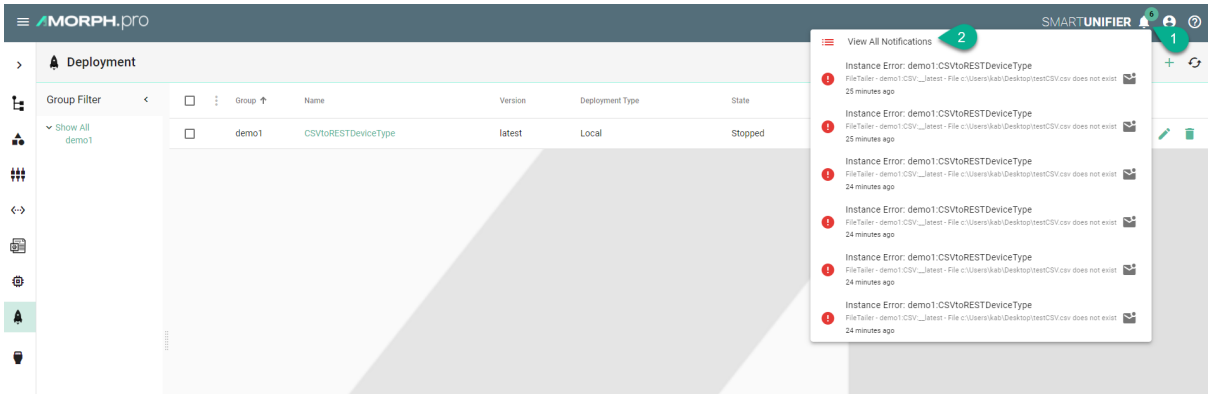


Select a notification **(3)** from the list and the **Dashboard (4)** will appear and display additional information.

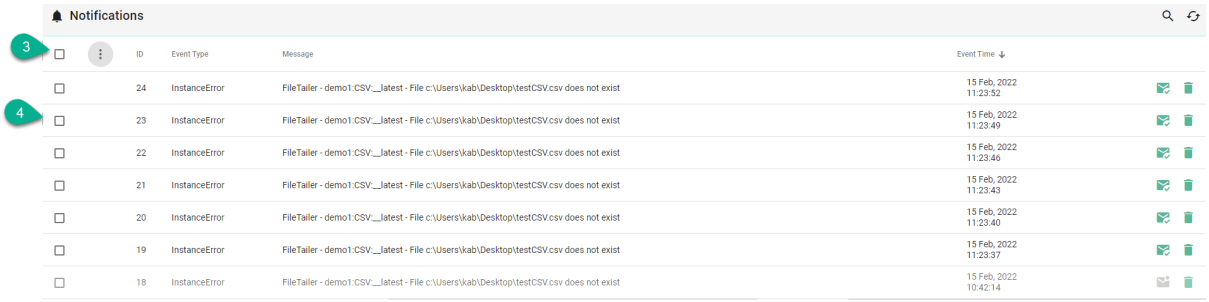


How to manage Notifications

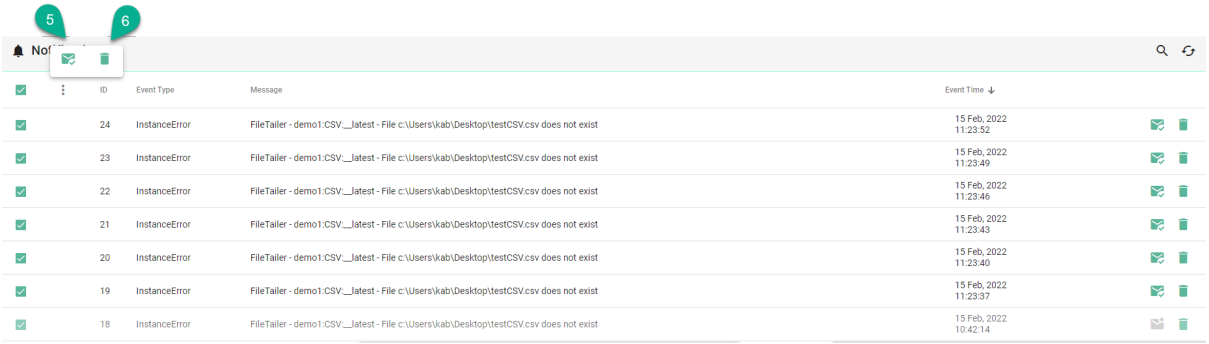
In order to manage the notifications click on the **Notifications** button **(1)** and select the **View All Notifications** option **(2)**.



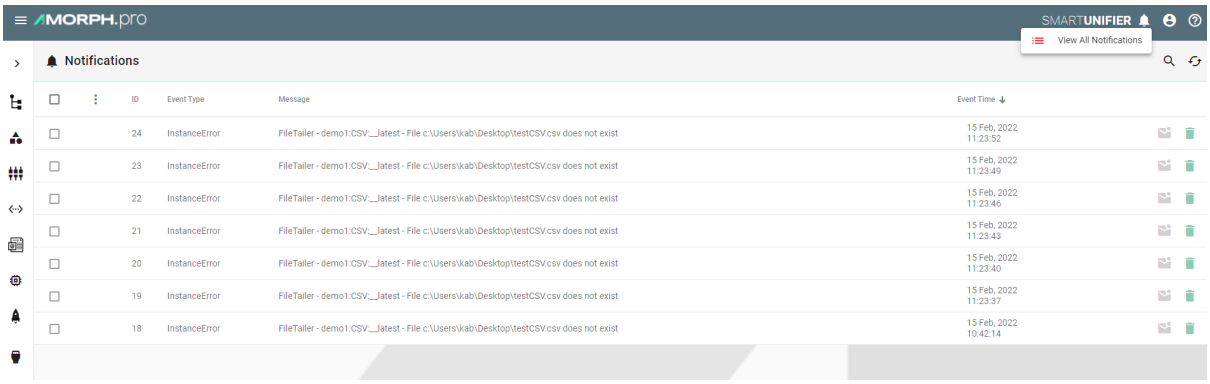
The Notifications Manager displays all the notifications. Select all **(3)** or specific notifications **(4)**.



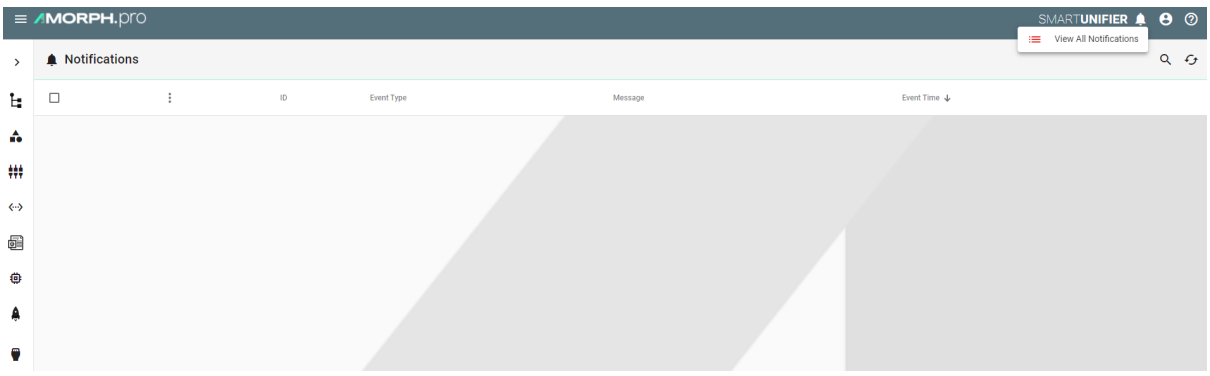
After selection a pop-up appears providing two options.



Click on the **Dismiss** button **(5)** to remove the selected notifications from the Notifications List. The selected notifications will still be available in the Notifications Manager.



To remove the selected notifications from the Notifications List and the Manager, click on the **Delete** button **(6)**.



CHAPTER

SIX

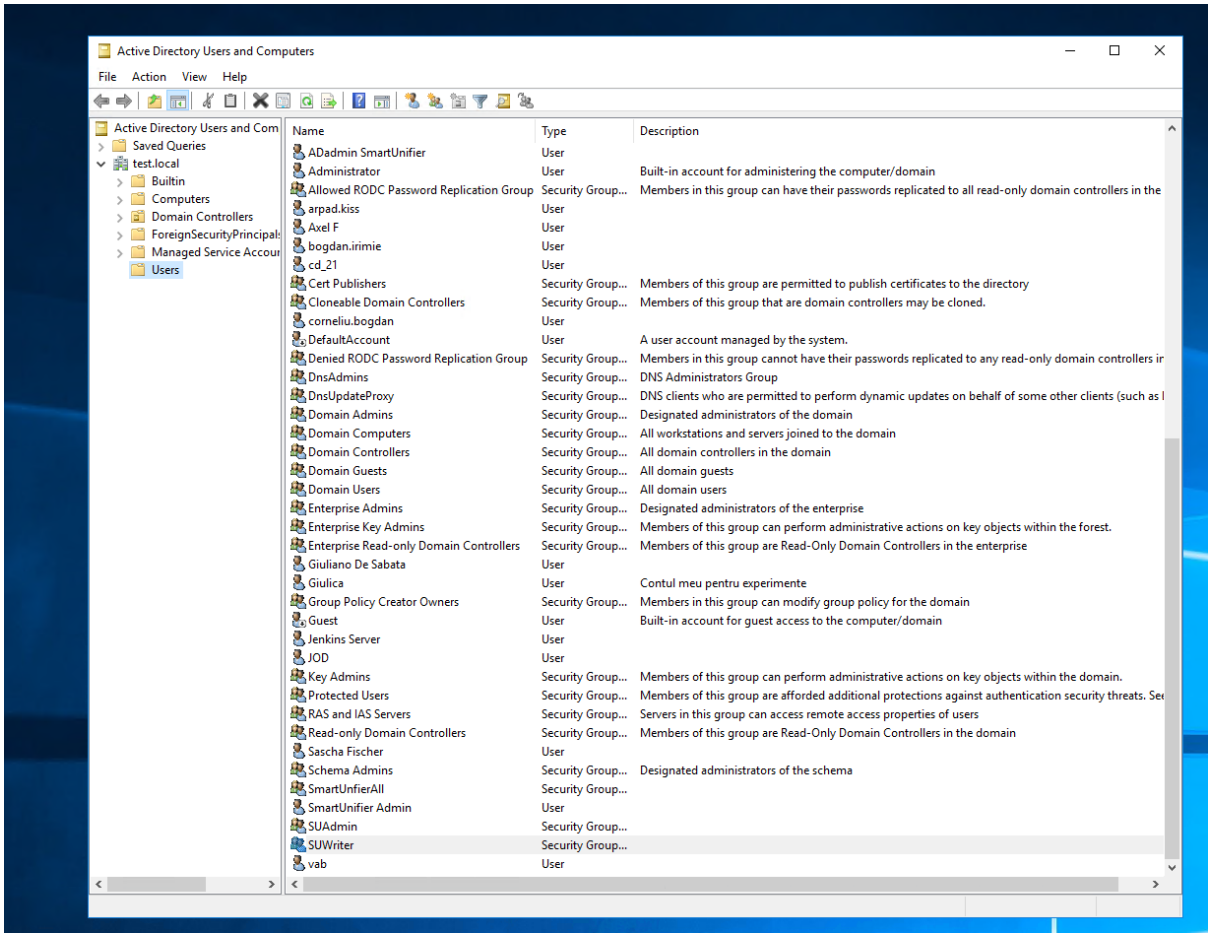
ADMINISTRATION

Learn how to:

- Integrate an *Active Directory*
- *Backup* and *Restore* the Repository
- Manage *Communication Channel Types*
- Manage *Docker Java Images*
- Create *Deployment Endpoints*
- Manage *Credentials*
- Manage *User Accounts*
- Manage *Logging Configurations*
- Create *Alerts*
- Manage *Alert Channels*
- Use *Extensions*
- Use *Environment Variables*
- Validate the *Configuration Components*

Active Directory Integration (ADI)

SMART**UNIFIER** supports Windows Active Directory (AD). System administrators can use the Active Directory to add/remove users, groups, and resources quickly and efficiently through one dashboard.



AD Group Mapping

An user from AD must be added to a group that acts as a role. The role determines what permissions are assigned to the user.

The mapping between the AD groups and the SMART**UNIFIER** roles is defined in the **application.conf** file from the **conf** folder.

application.conf

```
# https://www.playframework.com/documentation/latest/Configuration
include "default.conf"
apiPrefix = adapter

play = {
  server = {
    http.port = 9000
    http.address = "0.0.0.0"

    #http.port=disabled
    #https.port=9443
    #https.keyStore.path="path_to_keystore"
    #https.keyStore.password="keystore_password"
  }
  http.secret.key = "ChangeMySecret"
}
authentication {
        activedirectory {
            host = "192.168.0.132",
            port = 389,
            baseDN = "DC=test,DC=local",
            useSSL = false,
            user = "jenkins@test.local",
            password = "Aiurea05",
            groupmapping = {
                Administrator = "SUAdmin",
                Writer = "SUWriter",
                Reader = "SmartUnifierAll"
            }
        }
    }
unifiermanager {
  tempFolder = "temp"
  compiler {
    scala {
      javaHome = "jre"
    }
    management {
      dontDeleteWorkspace = true
    }
  }
  model {
    loadCodeFromScalaFile = false
  }
  deployment {
    javaHome = "jre"
    local = {
      deploymentFolder = "deploy"
      softRefreshInterval = 2000
      hardRefreshInterval = 5
      logStatusInterval = 30
      monitorLogs = true
    }
    docker {
      baseimage {
        autocreate = false
        jreImage = "adoptopenjdk/11-jre-hotspot"
      }
    }
  }
}
```

1

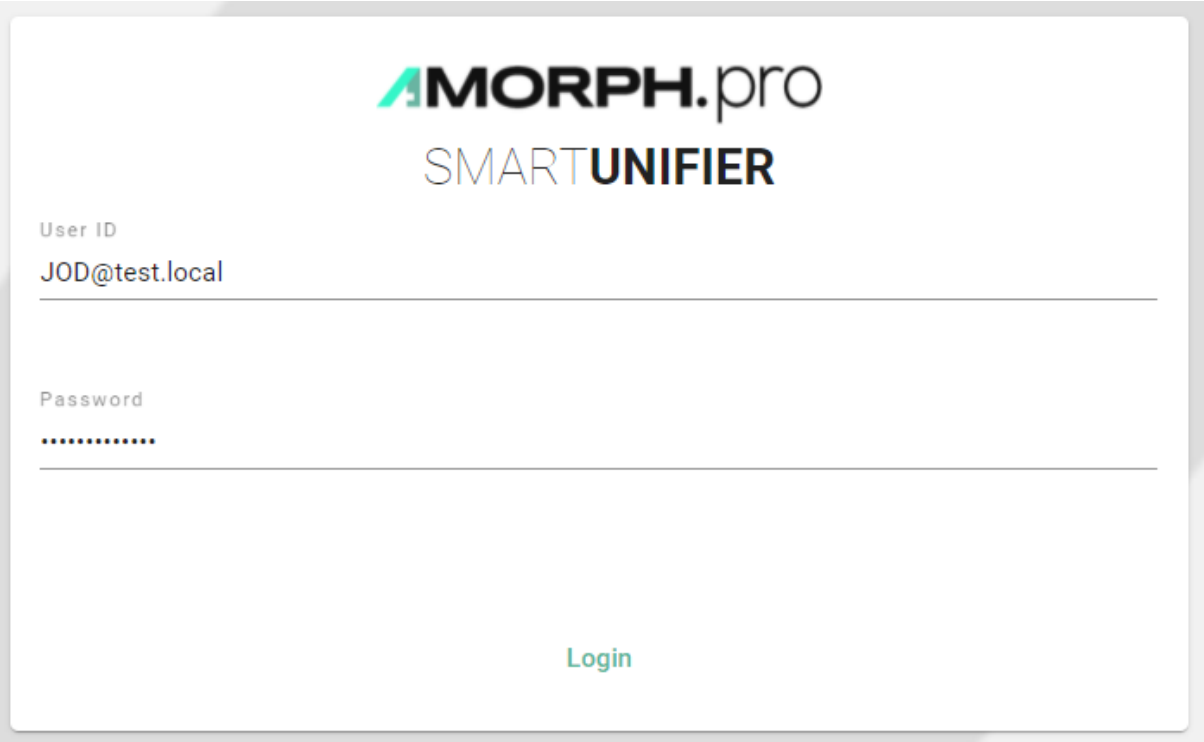
As seen above **(1)** in the left side are the SMART**UNIFIER** roles and in the right side, between the quotation marks are the AD groups.

The SMARTUNIFIER roles are predefined:

- Administrator - global permission
- Writer - limited permission, write and read access
- Reader - limited permission, read access

A user from an AD group will have permission based on the mapping of the AD group to a predefined SMART**UNIFIER** role.

After all the above configuration is done, the user can login to the SMART**UNIFIER** with the **User logon name** and the **Password** defined in AD.



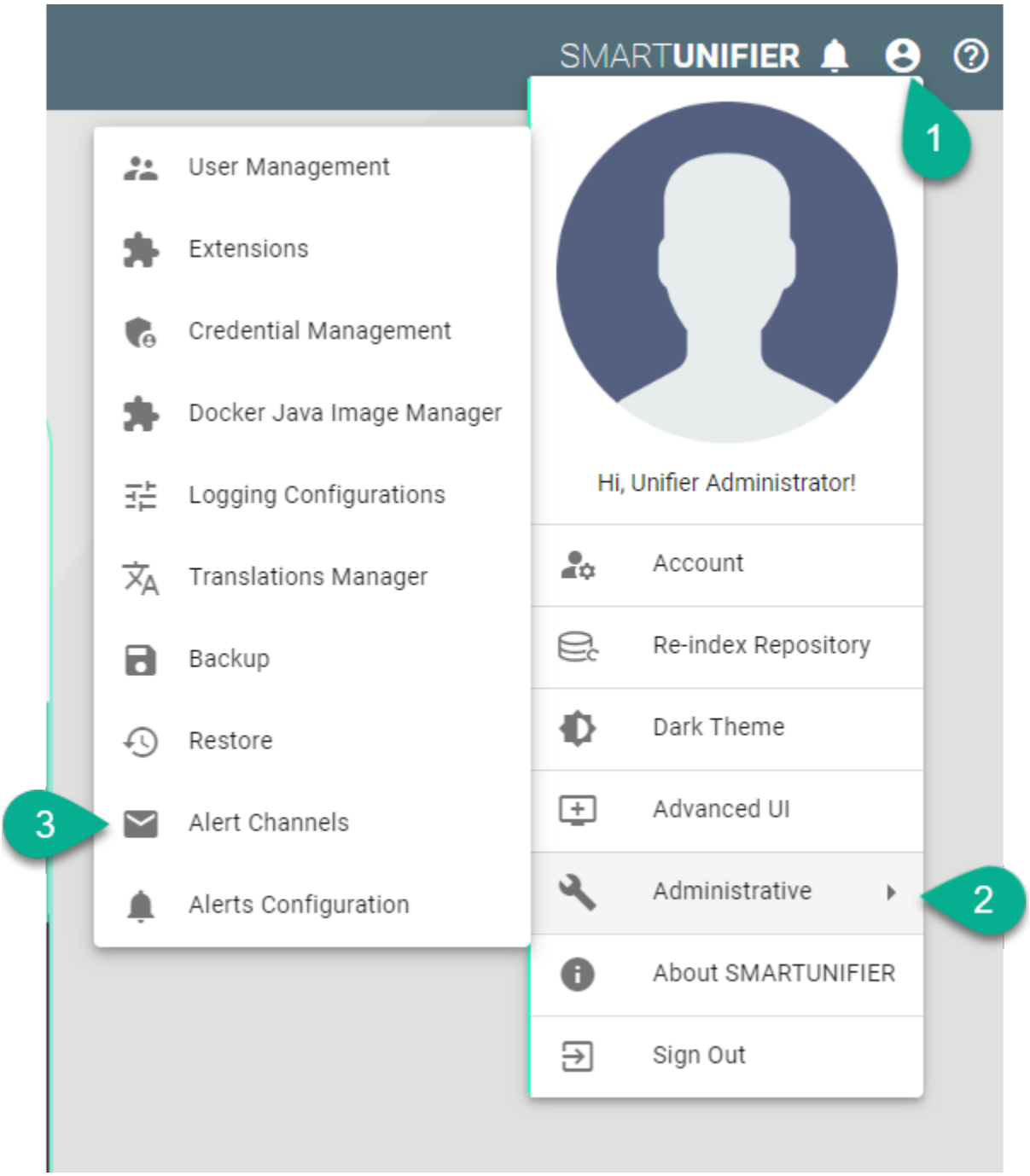
Alert Channels

Within the Alert Channels, the user can configure and manage the channels for sending alert notifications (e.g., send via email an alert for Instance errors).

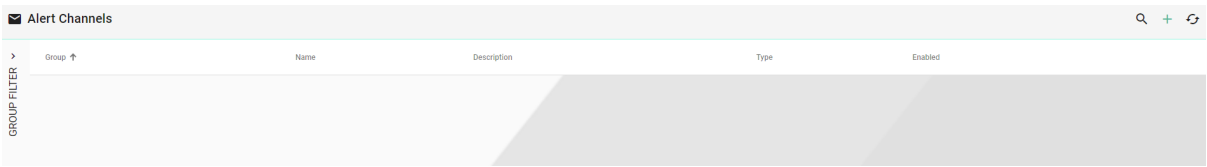
How to access

Follow the steps bellow to access the Alert Channels:

- Click on the **Account** icon **(1)**, go to **Administrative** section **(2)** and select the **Alert Channels** option **(3)**.



- The Alert Channels section is visible.



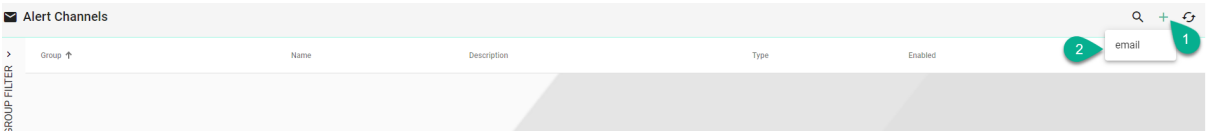
Note

The Alert Channels can only be accessed by user accounts with an administrator role assigned.

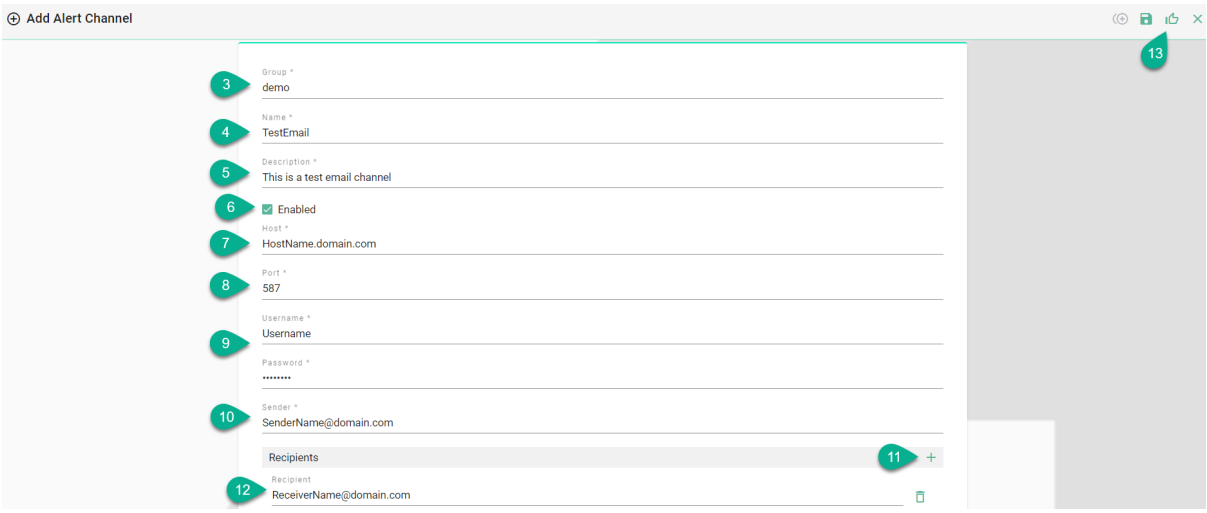
Add an Email Channel

Follow the steps described below to add an Email channel:

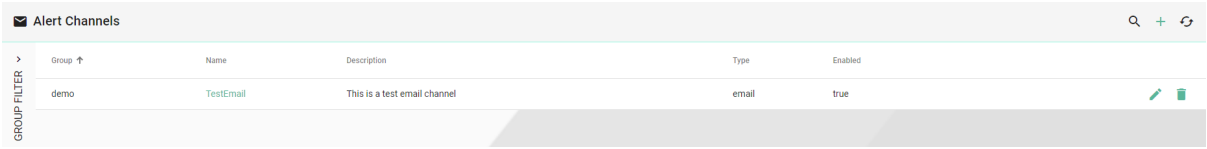
- Click on the **Add** button **(1)**.
- Select the **Email** option **(2)**.



- Type a name for the Group **(3)**.
- Input the email channel name **(4)**.
- Add description **(5)**.
- Click to check the **Enabled** box **(6)**.
- Provide the Host name **(7)**.
- Type the host Port **(8)**.
- Input the Username and Password **(9)**.
- Provide the Sender email address **(10)**.
- Click on the **Add** button **(11)** to input the Recipients email addresses **(12)**.

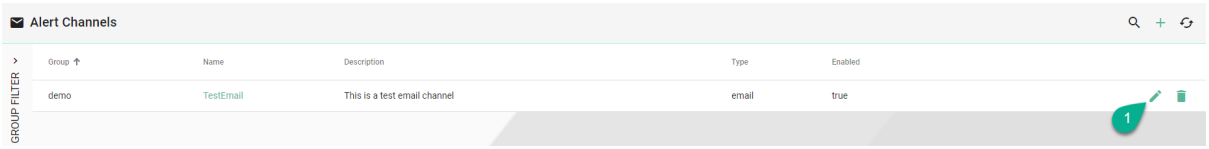


- Click on the **Save and Close** button **(13)**.

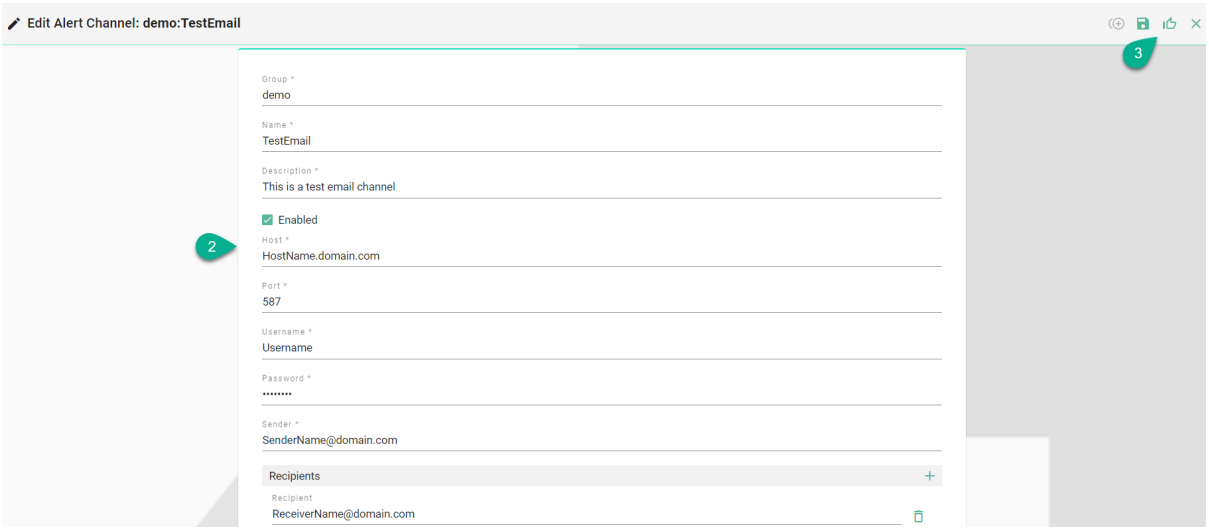


Edit Alert Channels

To edit an alert channel, select the **Edit** button **(1)**.

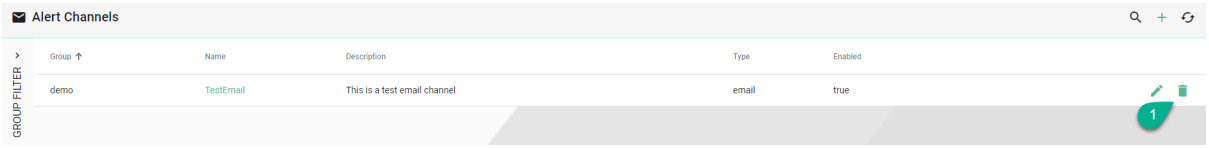


The Edit Mode is visible, the configuration can be edited **(2)** and then save the session by selecting the **Save and Close** button **(3)**.

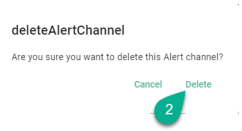


Delete Alert Channels

To delete an alert channel, select the **Delete** button **(1)**.



A pop-up confirmation appears, select the **Delete** button **(2)**.



Alerts Configuration

Within this section, the user can configure and manage alerts.

Alerts can be sent for:

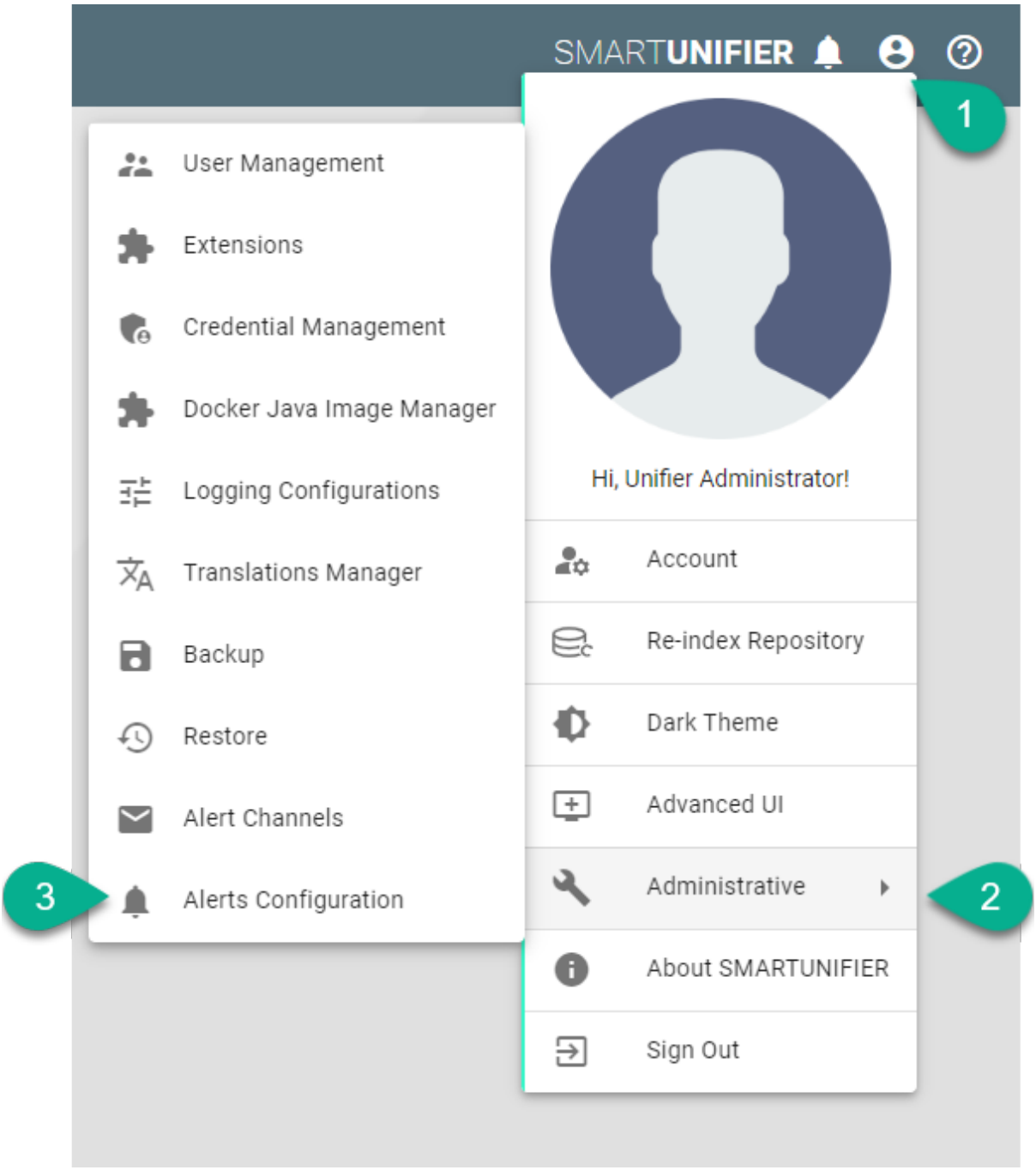
- Instance errors
- Instance Deployment status changed

- Endpoint Client status changed

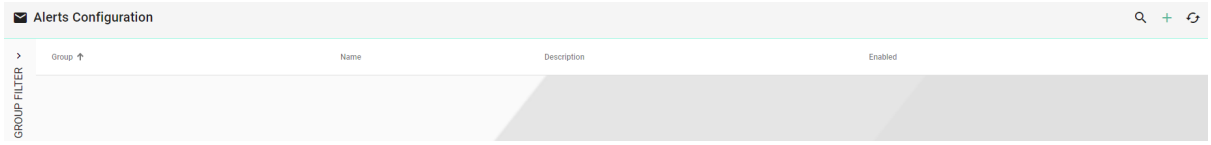
How to access

Follow the steps bellow to access the Alerts Configuration:

- Click on the **Account** icon **(1)**, go to **Administrative** section **(2)** and select the **Alerts Configuration** option **(3)**.



- The Alerts Configuration is visible.

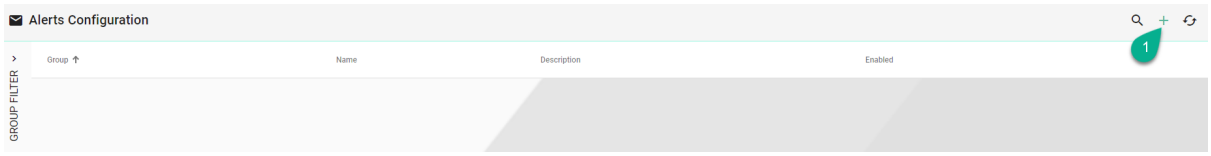


> **Note**
>
> The Alerts Configuration can only be accessed by user accounts with an administrator role assigned.

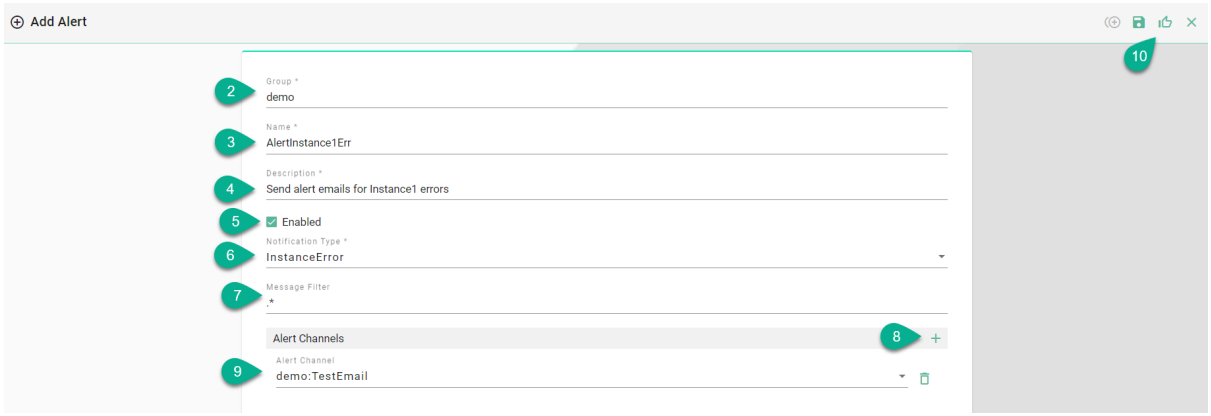
Add Alerts

Follow the steps described below to add an alert:

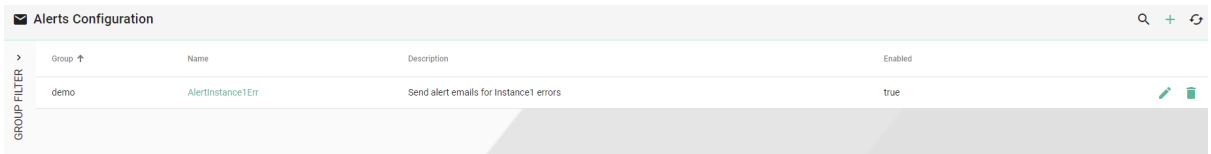
- Click on the **Add** button **(1)**.



- Type a name for Group **(2)**.
- Input the alert name **(3)**.
- Add description **(4)**.
- Check the box for Enabled **(5)**.
- Select the Notification Type **(6)**.
- Input the regex filter **(7)**.
- Click on the **Add** button **(8)** to select the alert channel **(9)**.

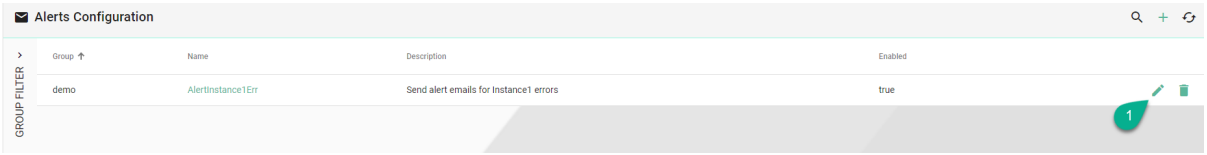


- Click on the **Save and Close** button **(10)**.

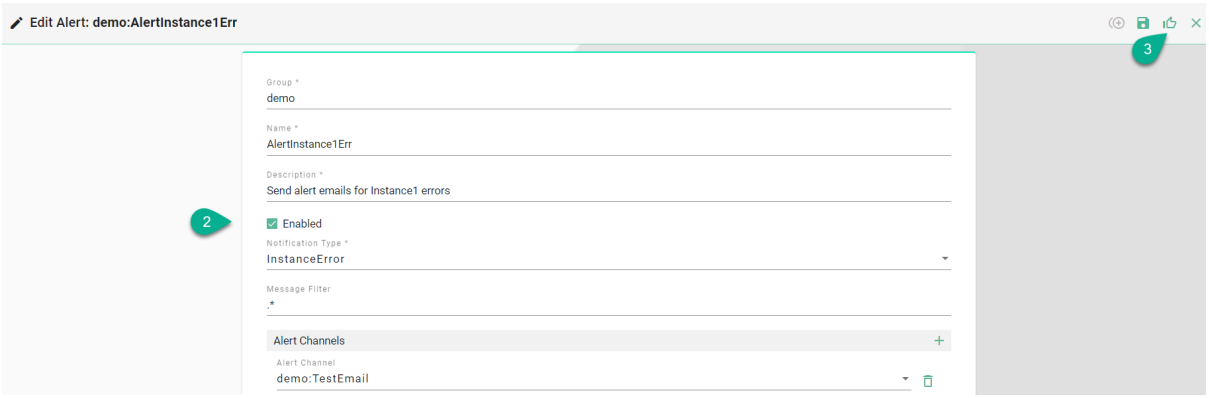


Edit Alerts

To edit an alert, select the **Edit** button **(1)**.

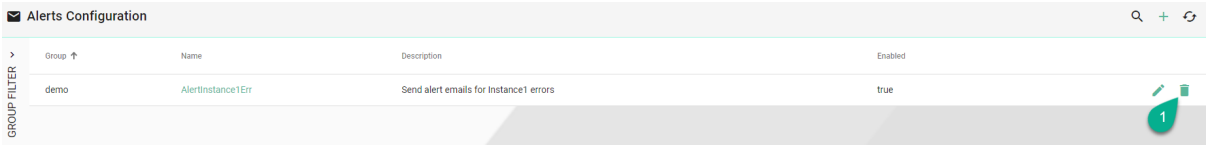


The Edit Mode is visible, the configuration can be edited **(2)** and then save the session by selecting the **Save and Close** button **(3)**.

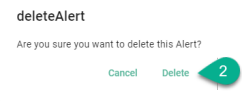


Delete Alerts

To delete an alert, select the **Delete** button **(1)**.



A pop-up confirmation appears, select the **Delete** button **(2)**.



Backup and Restore

SMART**UNIFIER** provides the possibility to manually backup and restore the repository and the internal database.

The **repository** represents a central location in which all the configuration components are stored:

- Information Models
- Communication Channels
- Mappings

- Device Types
- Communication Instances

The **internal database** is used for the operation of the SMART**UNIFIER** Manager and stores information like:

- User Accounts and Credentials
- Deployment Endpoints
- Base Images
- Channel Types
- Alert Configurations and Channels

How to access

To access the Backup or the Restore option, click on the **Account** icon **(1)**, go to the **Administrative** option **(2)** and select **Backup (3)** or **Restore (4)**.

Note

The Backup and the Restore features can only be accessed by user accounts with an administrator role assigned. Also keep in mind that the same SMART**UNIFIER** Manager version must be used.

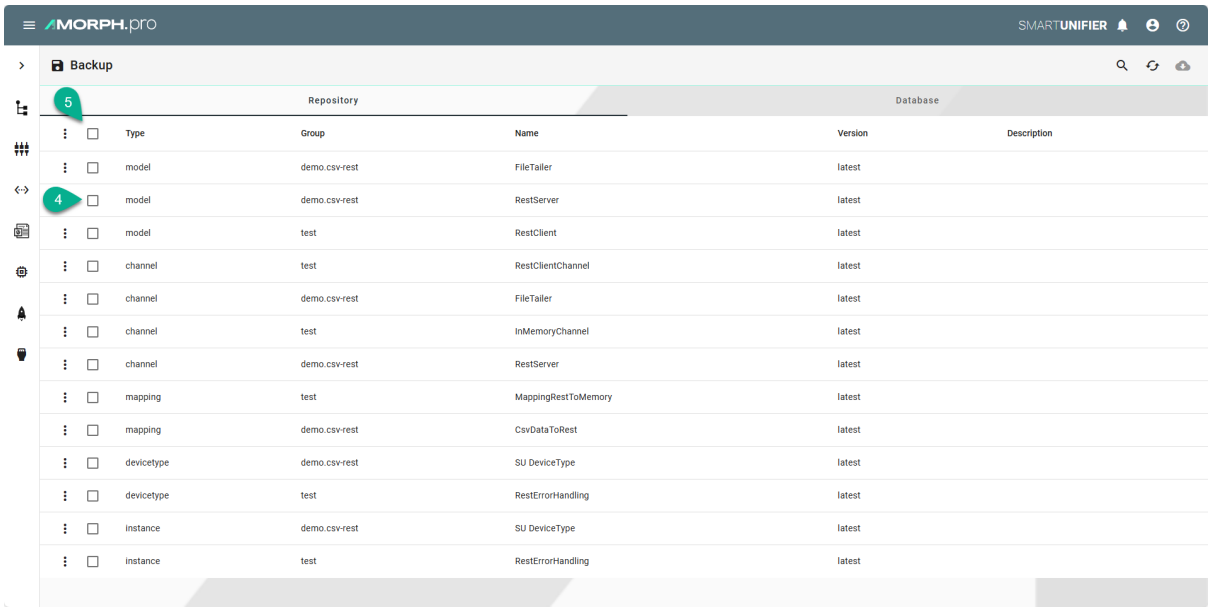
Backup

The Backup feature provides the possibility to create a copy of the configuration components to store elsewhere, so that it can be used to restore the last used after a data loss event occurs.

Follow the steps described below to create a backup of the repository:

- Select the **Account** icon **(1)**, go to the **Administrative** section **(2)** and select the **Backup** option **(3)**.

- The configuration components (Repository) are visible. Check the boxes **(4)** to select what to backup or check the top box **(5)** to select all.



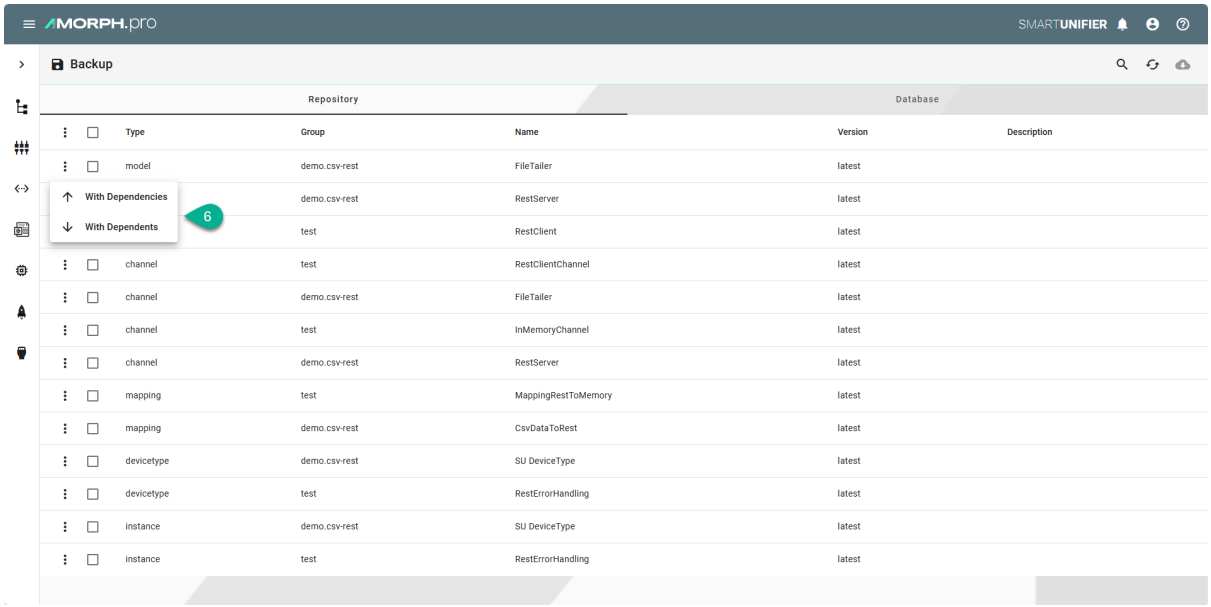
- If the selected component has dependencies or dependents, click on the three dots and select either one **(6)** to select or deselect all.

Note

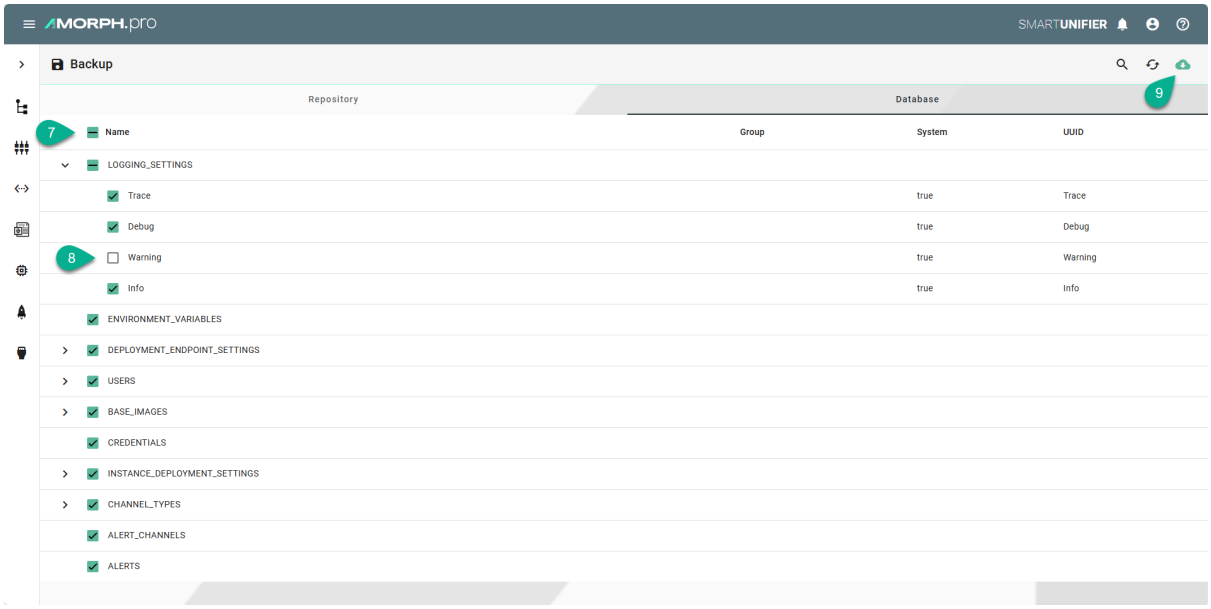
Press Ctrl + left Mouse Click to select/deselect all dependencies.

Note

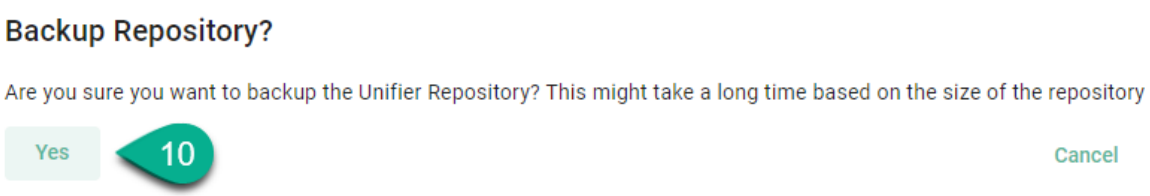
Press Ctrl + Shift + left Mouse Click to select/deselect all dependents.



- Click on the **Database** tab, check the top box to select all **(7)** or select the desired tables for backup. It is also possible to deselect some tables **(8)**.



- After the desired components are selected, click on the **Backup** button **(9)**. Click on the **Yes** button **(10)** to confirm.



Restore

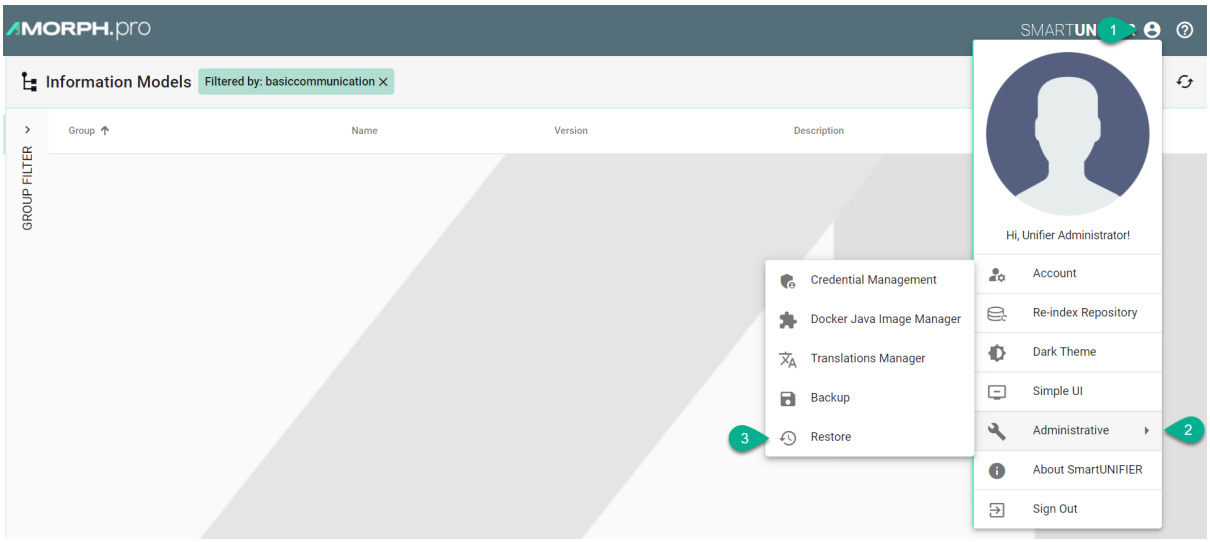
The Restore feature allows you to recover SMART**UNIFIER** configuration components from a backup and apply them to your current SMART**UNIFIER** Manager. This feature is particularly useful to share configuration components across several SMART**UNIFIER** Manager installations or when you encounter unexpected issues and need to revert back to a previous state.

> **Note**
>
> When restoring, the existing configuration components will be overwritten by with the selected configuration components from the backup if the name match!

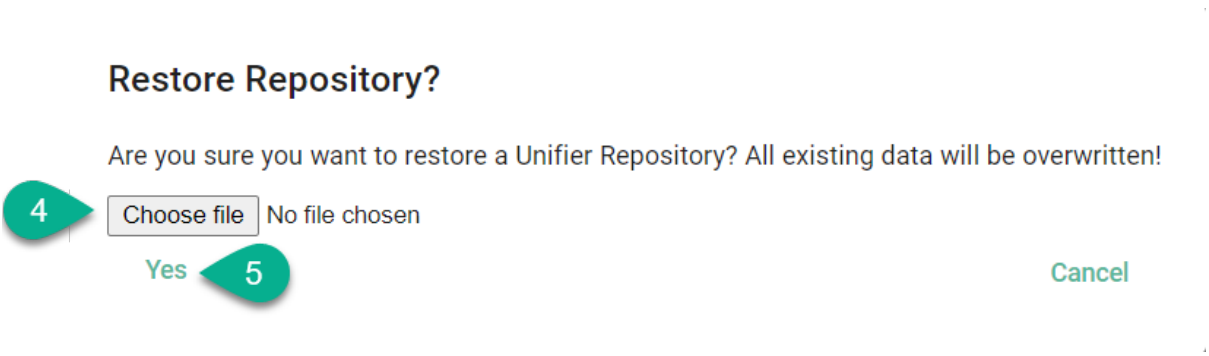
> **Note**
>
> Before restoring, ensure that you undeploy all communication instances.

Follow the steps described below to restore the SMART**UNIFIER** repository:

- Select the **Account** icon **(1)**, go to the **Administrative** section **(2)** and select the **Restore** option **(3)**.

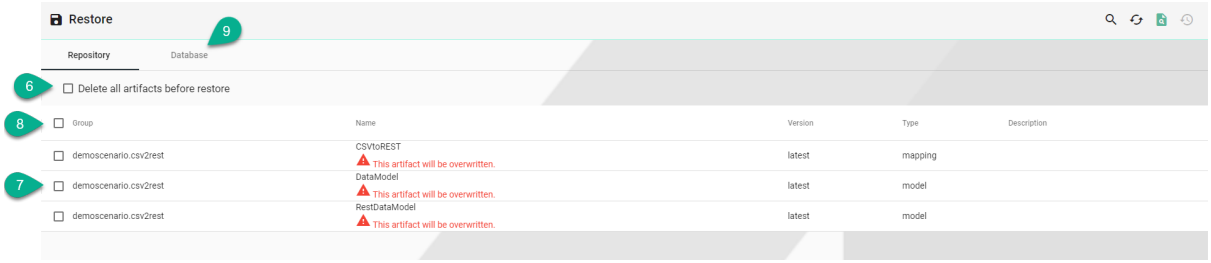


- A pop-up appears, choose the **TAR** file to restore **(4)** and select the **Yes** button **(5)** to confirm.



- The backup configuration components (Repository) are visible.

- If needed, check the box **(6)** to delete all existing components, before restoring.
- Check the boxes **(7)** to select what to restore or check the top box **(8)** to select all. Do the same for **Database** tab **(9)** if needed.



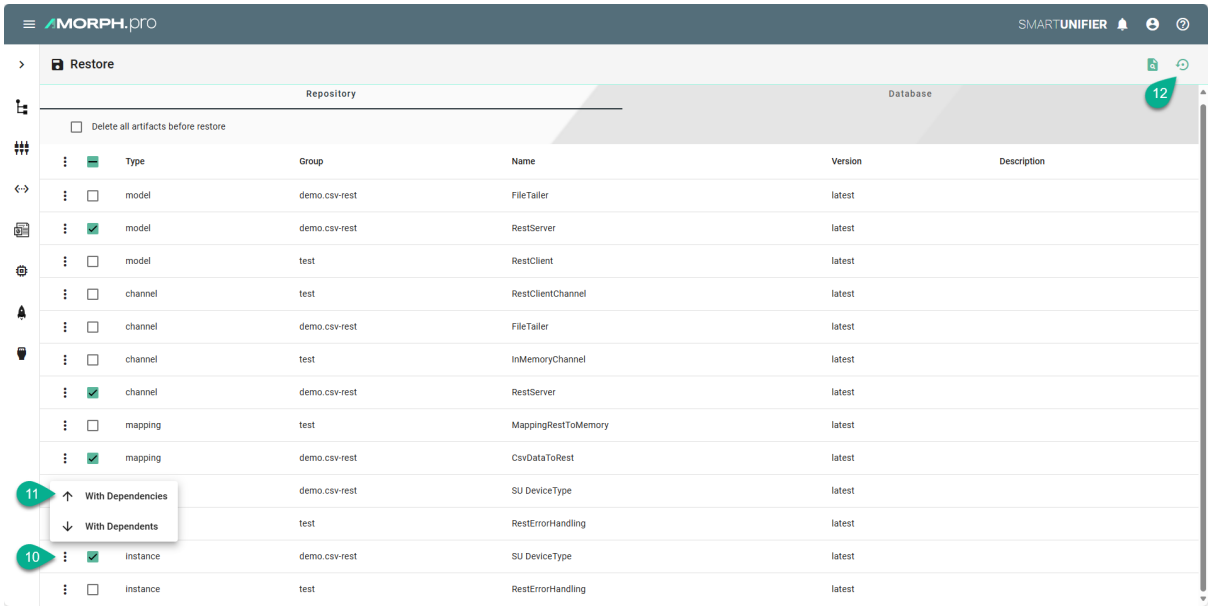
- If a component from the current configuration (if any) has the same name as one from the backup, it will be overwritten.
- If the selected component has dependencies or dependents, click on the three dots **(10)** and select either one **(11)** to select or deselect all.

Note

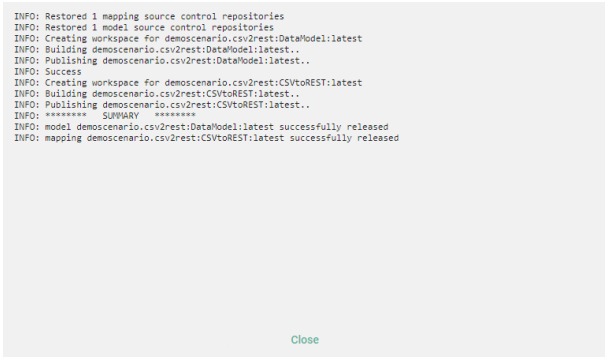
Press Ctrl + left Mouse Click to select/deselect all dependencies.

Note

Press Ctrl + Shift + left Mouse Click to select/deselect all dependents.



- After the desired components are selected, click on the **Restore** button **(12)**.
- The configuration components are uploading and all existing data will be overwritten!
- The uploading progress is displayed, including errors, if any.



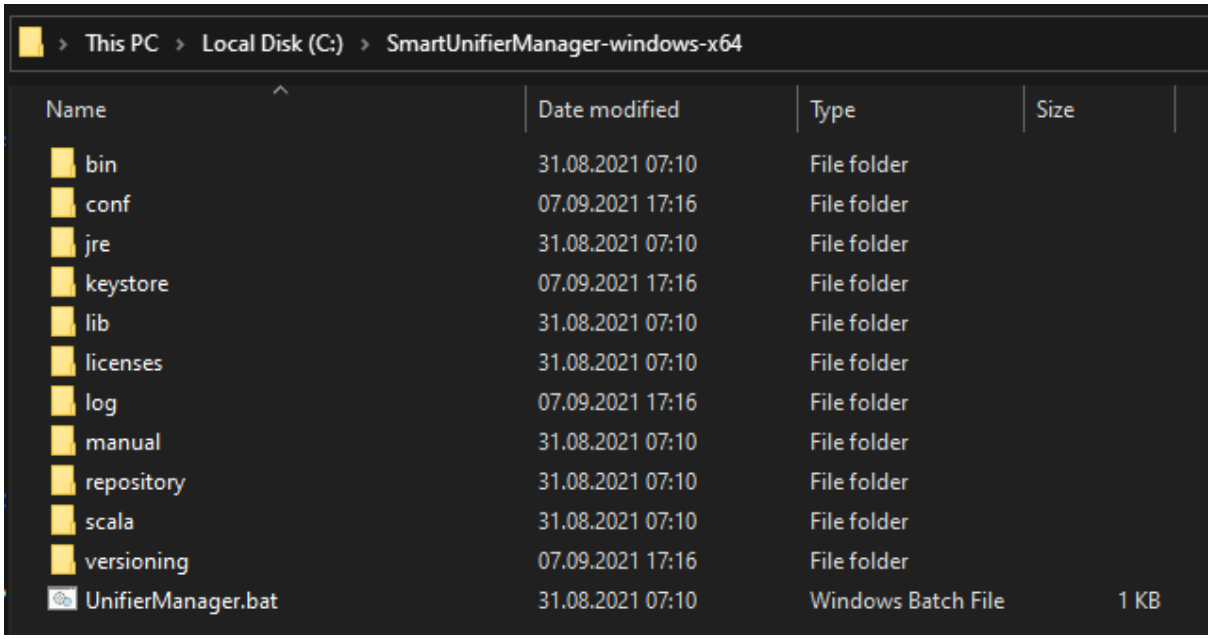
- Click on the **Close** button to finish.

Manager Backup

In order to backup SMART**UNIFIER** Manager make a copy of the SMART**UNIFIER** installation package.

Before the backup make sure to remove the following directories:

- temp
- workspace
- log
- deploy



Channel Types Manager

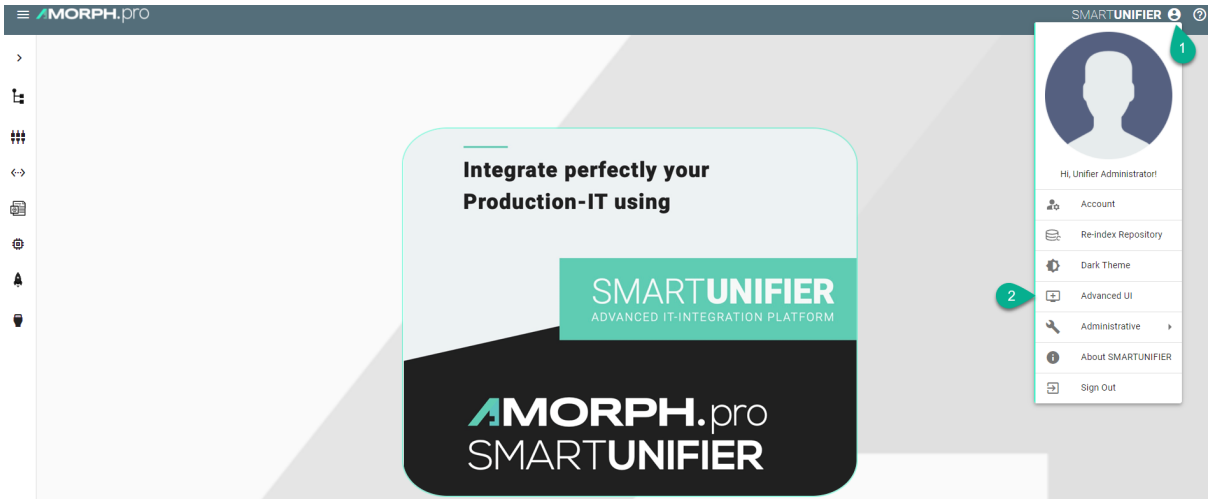
By default, the Channel Types Manager displays all *Channels* included in your current version of SMART**UNIFIER**.

Communication Channels that should be used within the configuration of a SMART**UNIFIER** Communication Instance have to exist in the Channel Types Manager. How to add new Channel Types is shown in the *section below*.

How to access

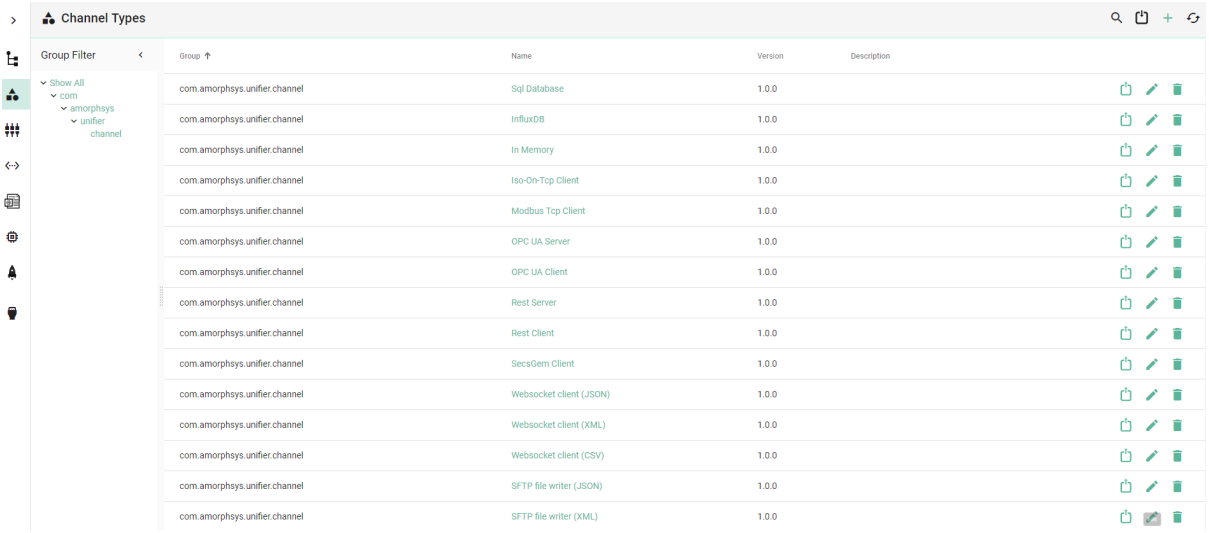
Follow the steps bellow to access the Channel Types Manager:

- Click on the **Account** icon **(1)** and select the **Advanced UI (2)**.



- Click on the **Channel Types** button **(3)** to open the Channel Types perspective.

- The main view of the Channel Types is visible.



Note

The Channel Types Manager can only be accessed by user accounts with an administrator role assigned.

About Layers

Implementations of SMART**UNIFIER** Communication Channels consist of one and up to three so-called layers.

The target of layers is to transform data from Information Models into the respective data format of the specific protocol used in case the data traffic is outgoing from a SMART**UNIFIER** Communication Instance. The same principle applies when data is incoming.

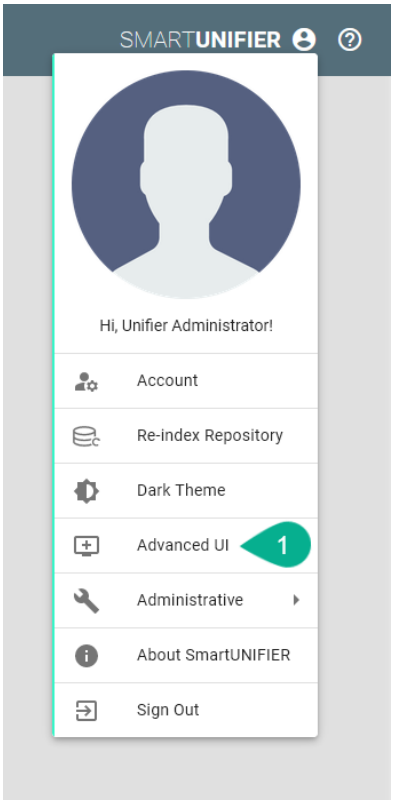
As an example for such a layer stack you can see below the layer stack for the *MQTT Communication Channel*:

- Data that is incoming from a Device is transformed into a String behind the scene.
- The String is then converted into a JSON Object.
- Finally, the JSON is used to assign data to the assigned Information Model.

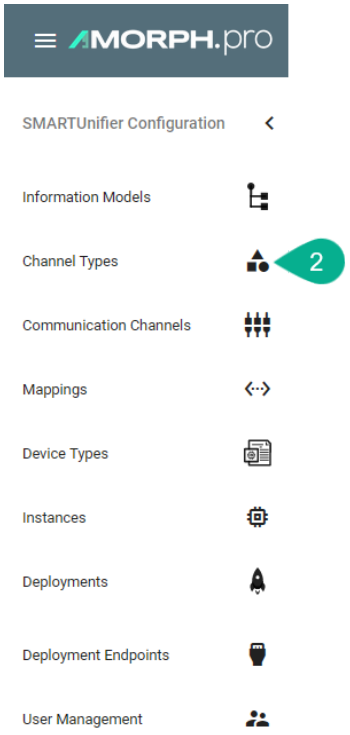
How to create a new Channel Type

Follow the steps below to create a new Channel Type:

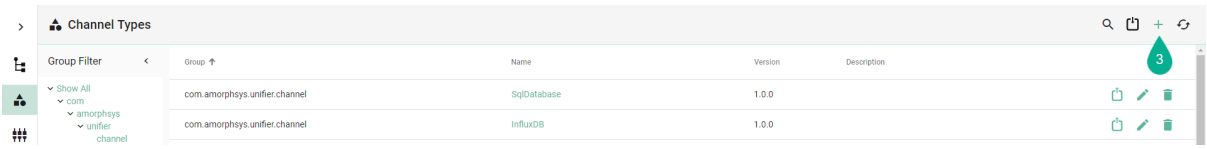
1. Open the SMART**UNIFIER** menu and select **Advanced UI**.



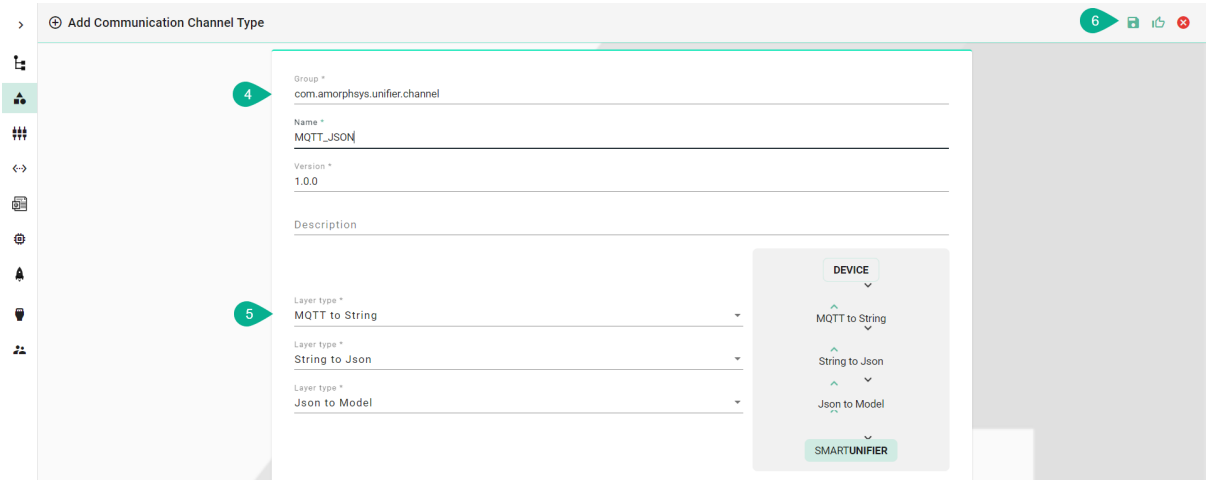
2. Go to the Channel Types perspective by clicking the **Channel Types** button.



3. Click on the **Add** button in the upper right corner.



4. Enter some descriptive information:
    - Enter a **group**
    - Enter the **name** of the Channel
    - Enter a **version**
5. Next, define the layer stack of the new Channel Type:
    - Select a layer with the **Layer type** drop-down menu.
    - In case the selected layer has more layers dependent on itself, select again another layer with the **Layer type** drop-down menu showing up below.
6. To save the Communication Channel Type select the **Save** button.



Command line interface (CLI)

Overview

SmartUnifierCli is a command-line tool designed for managing backups and restores in the Smart Unifier system. It is intended for use in automation scripts (`.bat` and `.sh`), making it suitable for programmatic execution.

Usage

Command Syntax

```
SmartUnifierCli <command> [options]
```

Available Commands

1. Backup System

Fetches backup data and saves it locally.

```
SmartUnifierCli backup -u <username> -p <password> [-f <filename>]
```

Options:

- `-u`: Username for authentication.

- `-p`: Password for authentication.
- `-f <filename>`: Optional. Specifies the filename to save the backup.

Example:

```
SmartUnifierCli backup -u admin -p secret -f my_backup.tar.gz
```

2. Restore System

Uploads a backup file for restoration.

```
SmartUnifierCli restore <file-path> -u <username> -p <password>
```

Example:

```
SmartUnifierCli restore /path/to/backup.tar.gz -u admin -p secret
```

The restore process outputs streamed progress information to the console.

Exit Codes

SmartUnifierCli returns an **exit code** of `0` when the command executes successfully. Any **non-zero exit code** indicates an error. This behavior allows integration into scripts where error handling is required.

Shell Script Example (`.sh`)

```
SmartUnifierCli backup -u admin -p secret -f my_backup.tar.gz
if [ $? -ne 0 ]; then
    echo "Backup failed"
    exit 1
fi
```

Windows Batch Script Example (`.bat`)

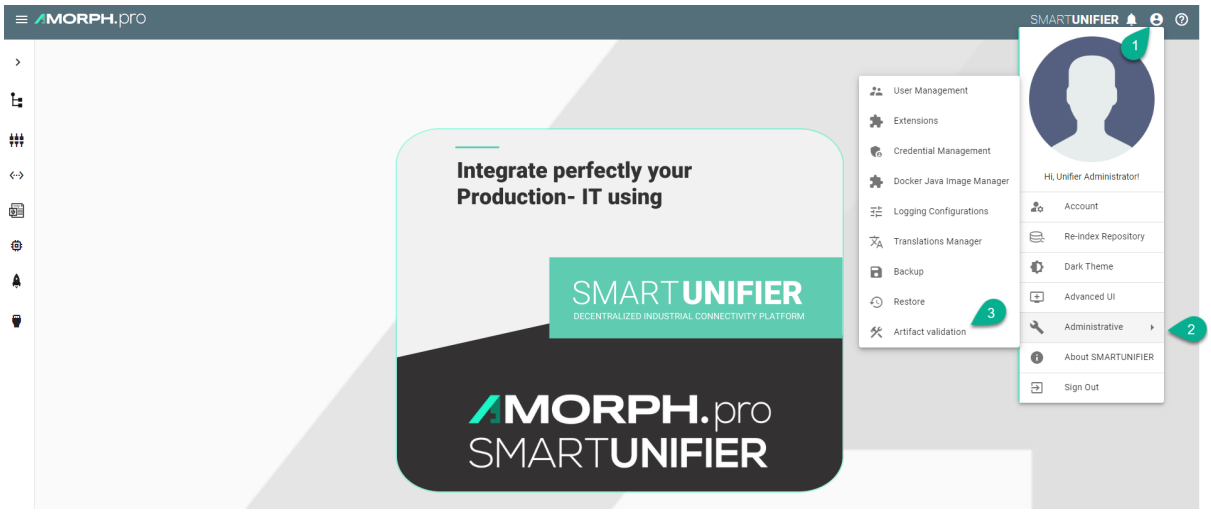
```
@echo off
SmartUnifierCli backup -u admin -p secret -f my_backup.tar.gz
if %ERRORLEVEL% NEQ 0 (
    echo Backup failed
    exit /b 1
)
```

Configuration Components Validation

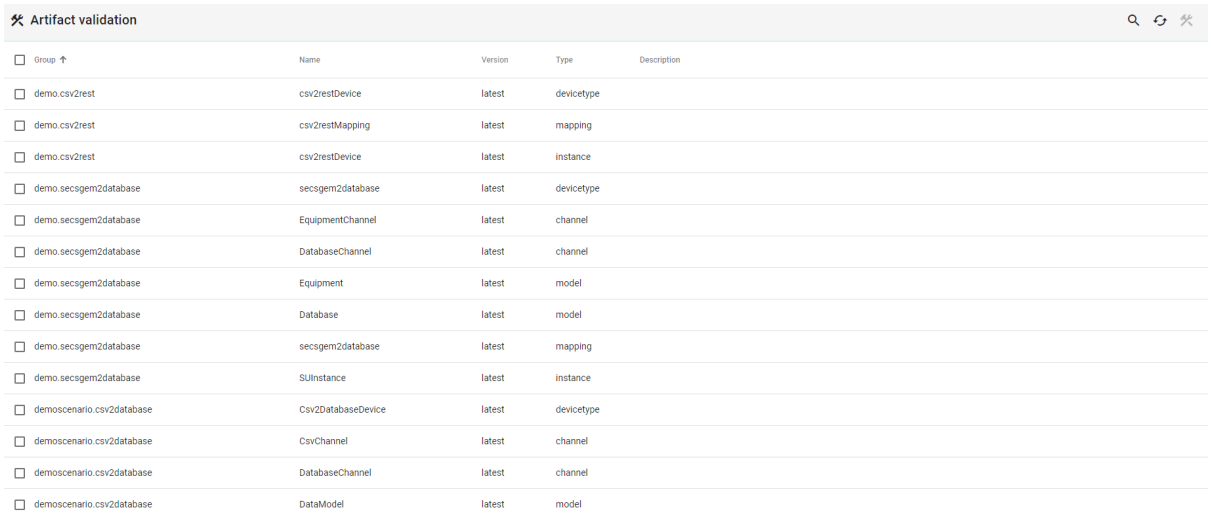
SMART**UNIFIER** provides the possibility to validate configuration components (artifacts) that are created. This is especially important when restoring a backup from an older version of a SMART**UNIFIER** Manager installation.

How to access

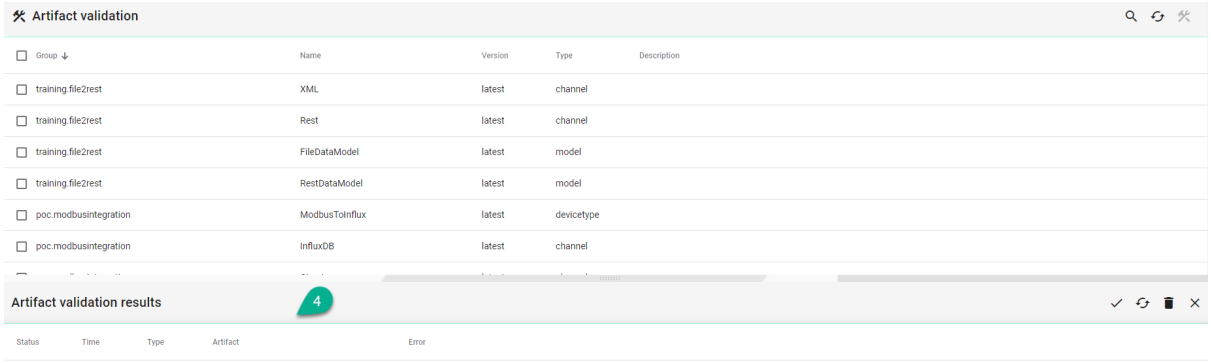
To access the Artifact Validation option, click on the **Account** icon **(1)**, go to the **Administrative** option **(2)** and select the **Artifact Validation** perspective **(3)**.



The Artifact Validation perspective is visible, displaying all the configuration components.



Press **F8** from the keyboard to open the **Artifact Validation Results** view **(4)**.



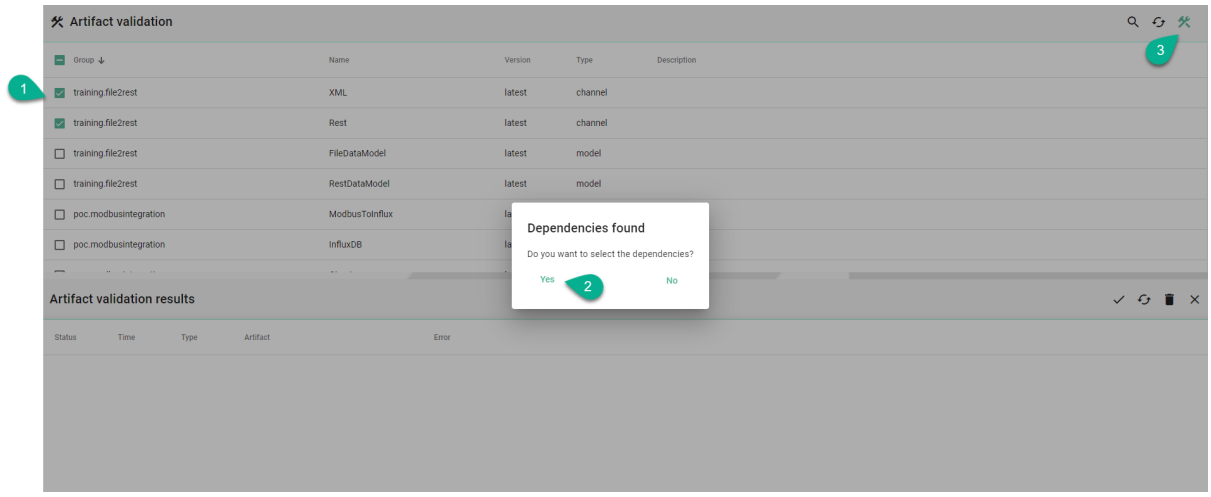
Note

The Artifact Validation features can only be accessed by user accounts with an administrator role assigned.

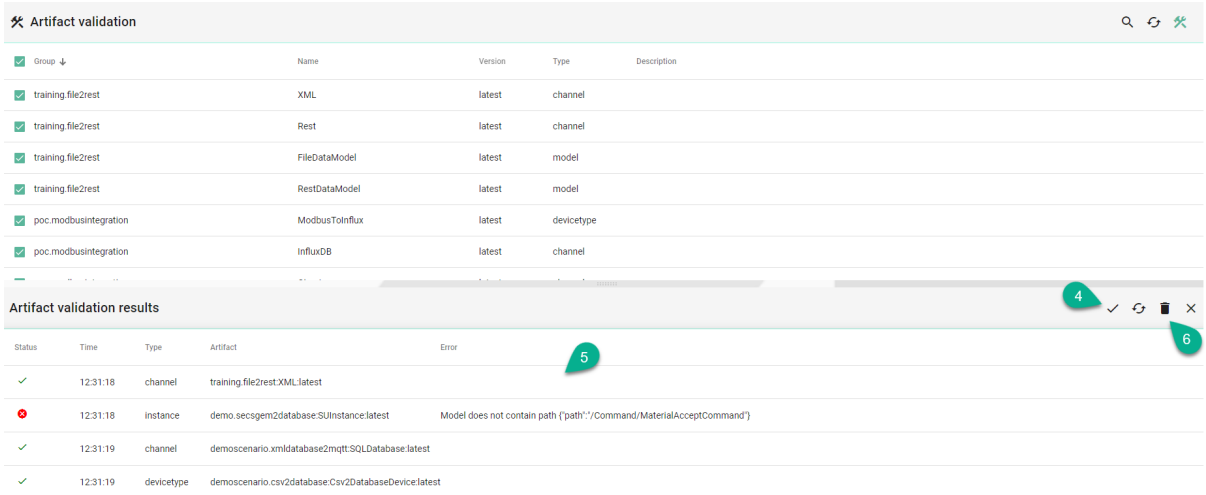
How to Validate Artifacts

Follow the steps described below to validate artifacts:

- From the Artifact Validation perspective select the desired artifacts **(1)**
- If dependencies are found, a pop-up will appear. Click on the **Yes** button **(2)** to select the dependencies
- Click on the **Validate** button **(3)**



- Make sure the **Artifact Validation Results** view is opened and click on the **Show valid** button **(4)** to see the results **(5)**



- If an artifact is valid, the **Status** column will show a green valid icon
- If an artifact is NOT valid, the **Status** column will show a red **X** icon and the **Error** column will provide details
- To delete the validation results, click on the **Clear** button **(6)**

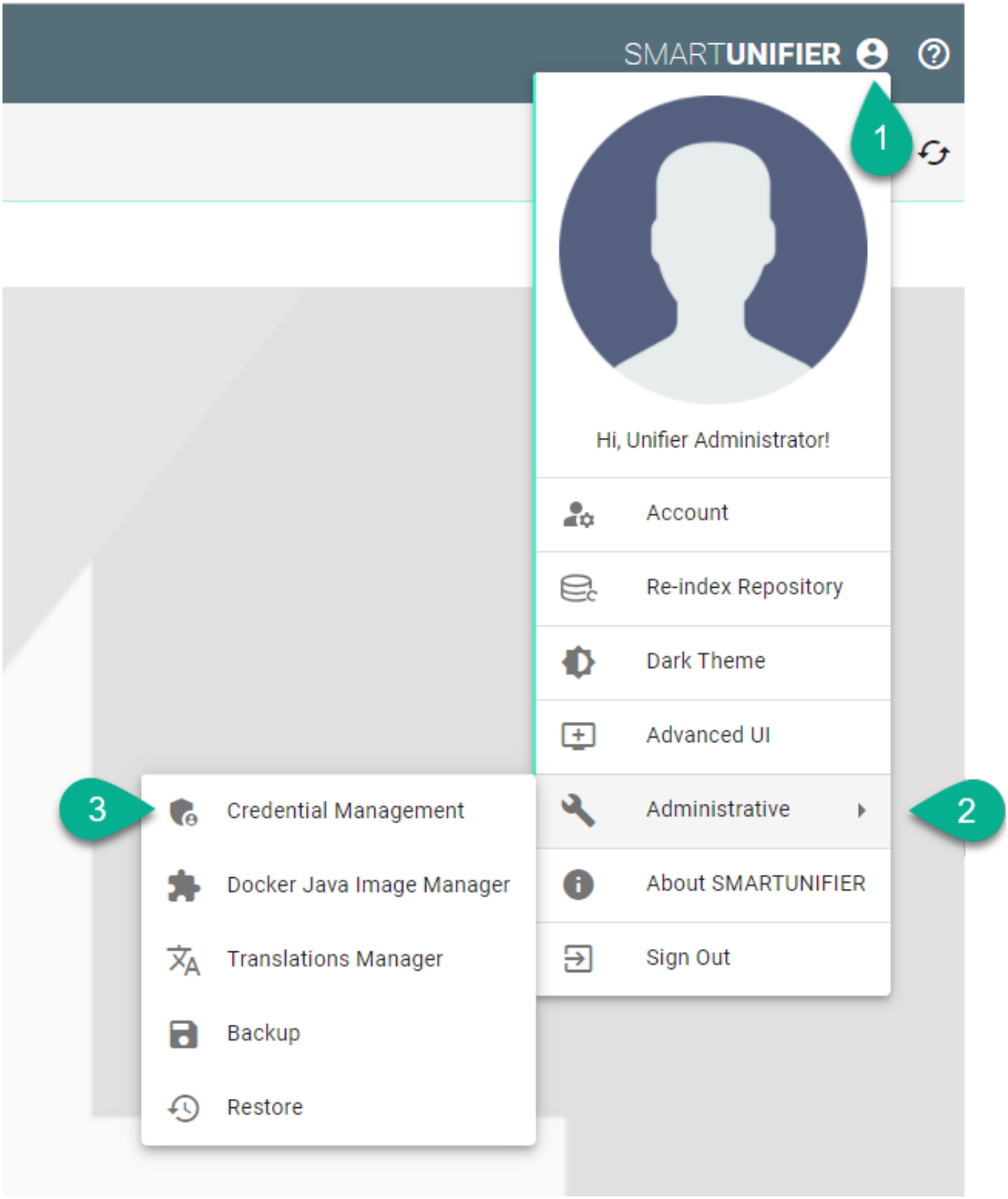
Credential Management

Within the Credential Manager the user can store and manage the credentials needed for the Communication Channel configuration (e.g., password for certificates, username and password for SQL Server).

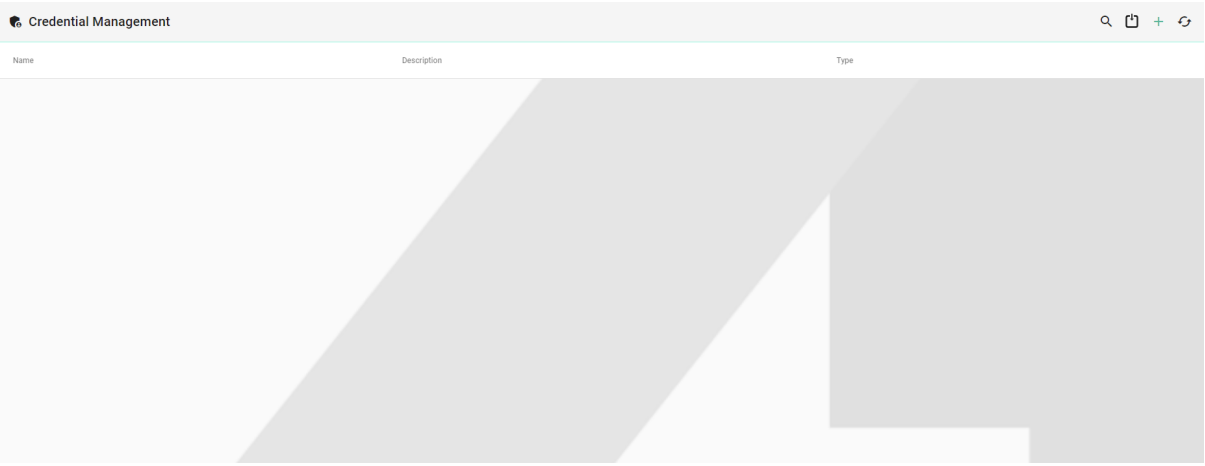
How to access

Follow the steps bellow to access the Credential Management:

- Click on the **Account** icon **(1)**, go to **Administrative** section **(2)** and select the **Credential Management** option **(3)**.



- The Credential Management is visible.



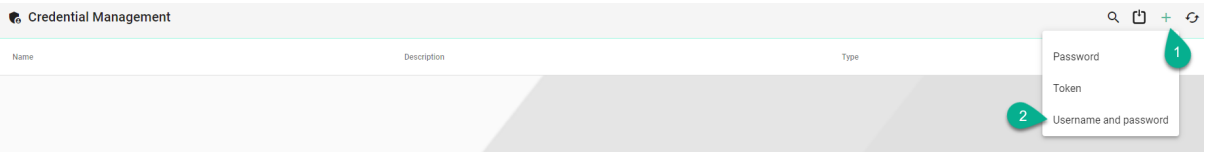
Note

The Credential Management can only be accessed by user accounts with an administrator role assigned.

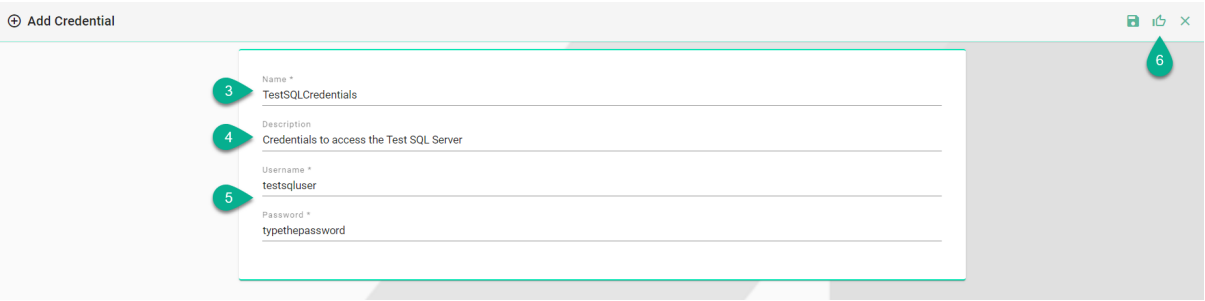
Add Credentials

Follow the steps described below to add credentials:

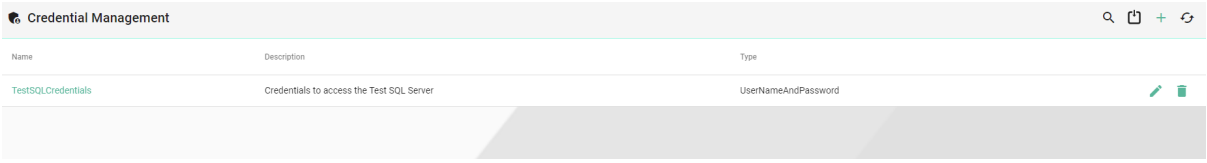
- Click on the **Add** button **(1)**.
- Select an option **(2)** Password or Username and Password.



- Type a name for Credentials **(3)**.
- Add description **(4)** (optional).
- Input the Username and Password **(5)**.



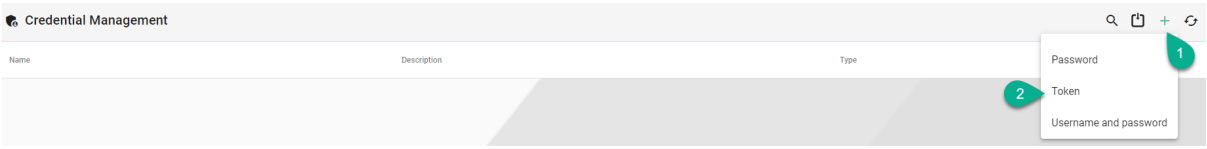
- Click on the **Save and Close** button **(6)**.



Add Token

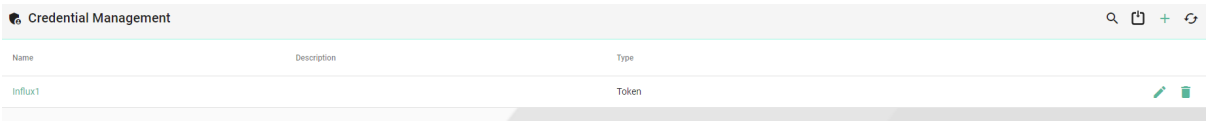
Follow the steps described below to add a token:

- Click on the **Add** button **(1)**.
- Select the **Token** option **(2)**.



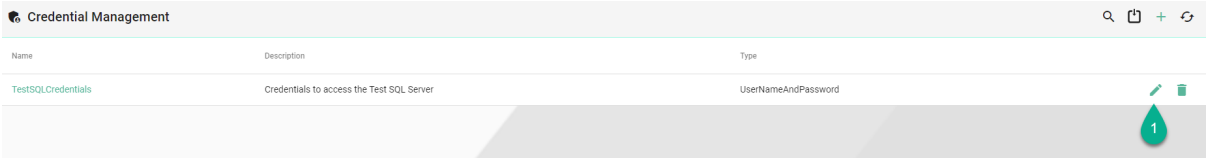
- Type a name for the Token **(3)**.
- Add description **(4)** (optional).
- Input the Token **(5)**.

- Click on the **Save and Close** button **(6)**.

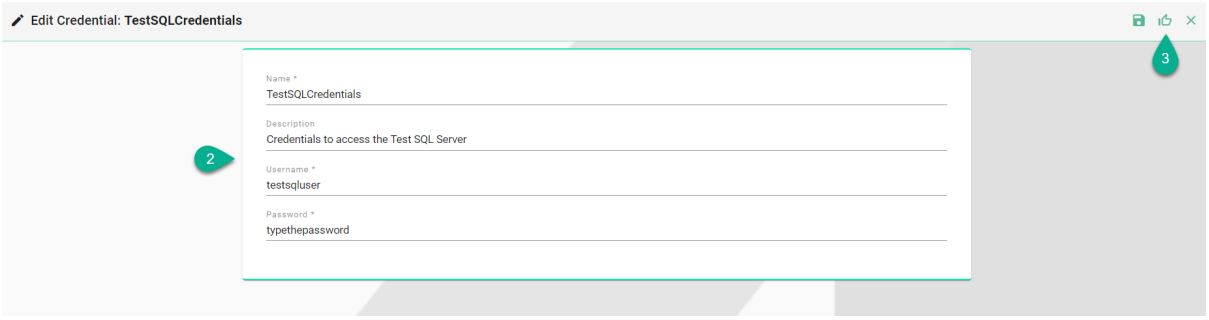


Edit Credentials

To edit the credentials, select the **Edit** button **(1)**.

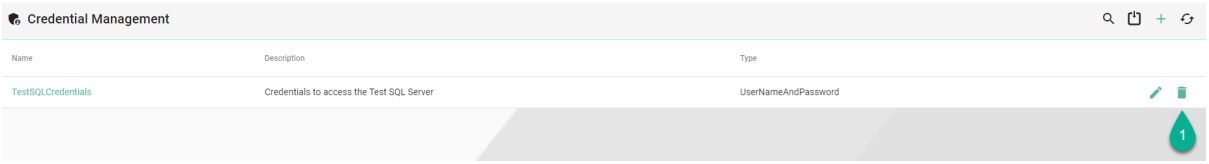


The Edit Mode is visible, the configuration can be edited **(2)** and then save the session by selecting the **Save and Close** button **(3)**.

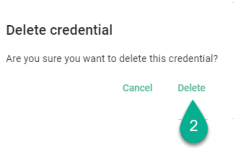


Delete Credentials

To delete credentials, select the **Delete** button **(1)**.



A pop-up confirmation appears, select the **Delete** button **(2)**.

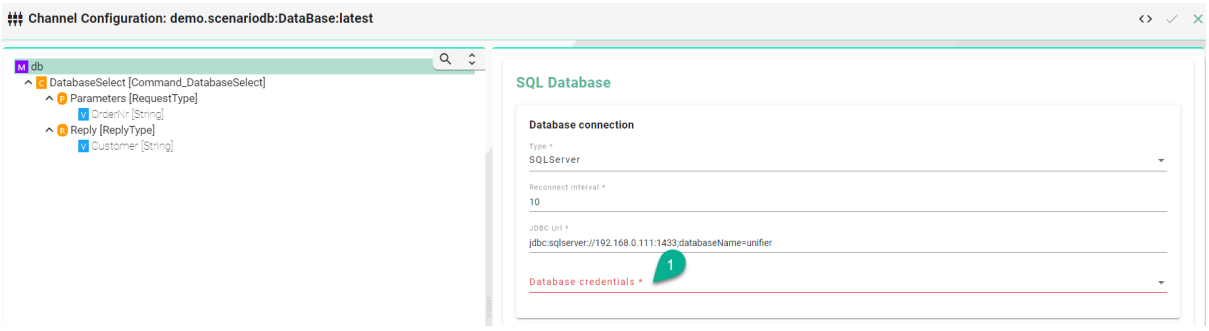


Using Credential Manager when configuring the Communication Channels

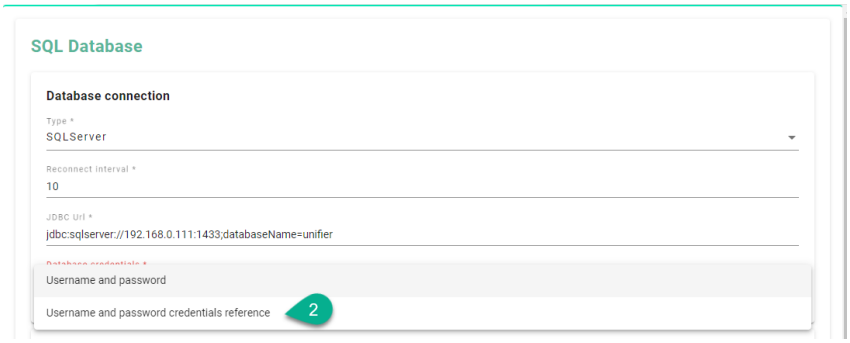
When configuring the Communication Channels, the user has the option to manually input the credentials or to select one from the Credential Manager.

Example of SQL Database Communication Channel configuration:

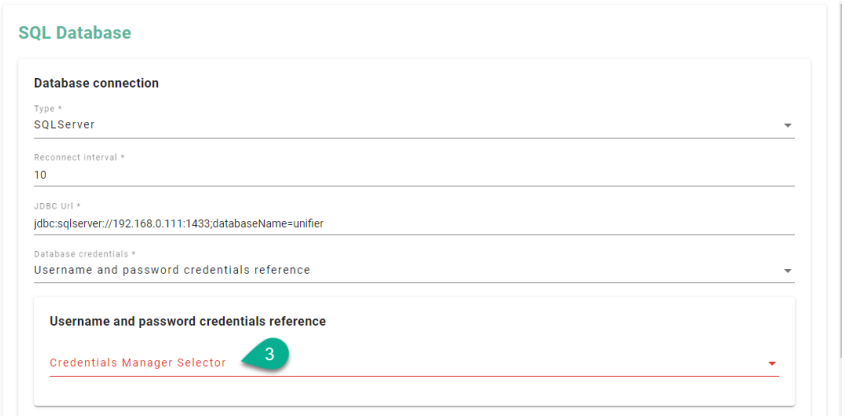
- Click on the **Database credentials** field **(1)**.



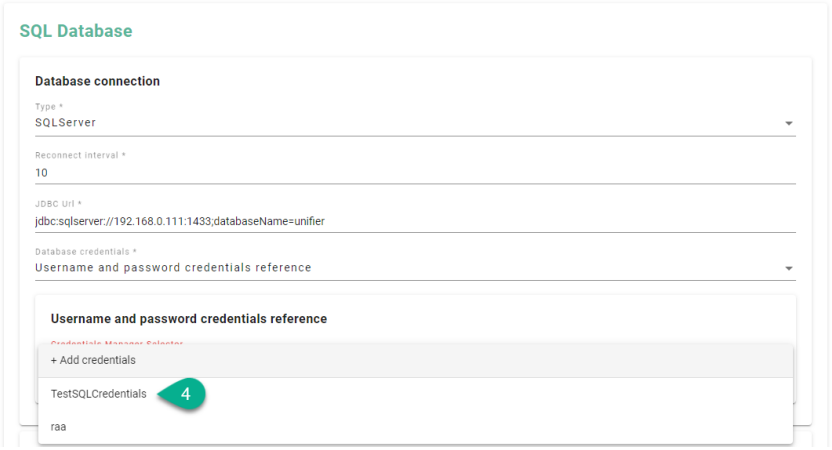
- Select the **Username and password credentials reference** option **(2)**.



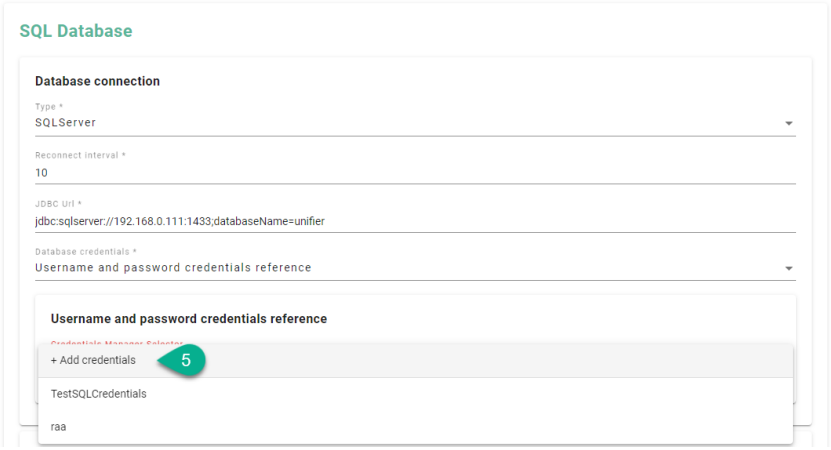
- Click on the **Credentials Manager Selector** option **(3)**.



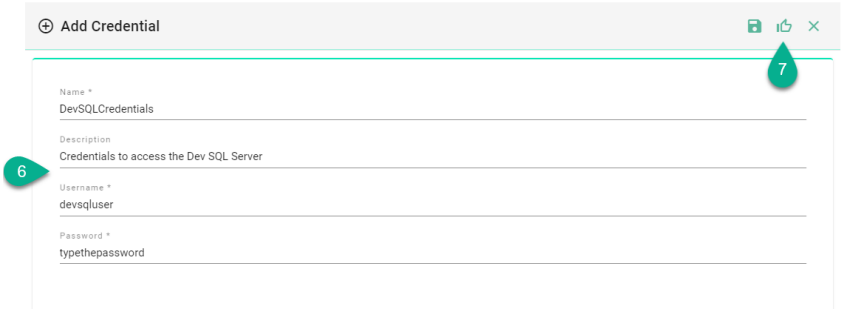
- Select one of the credentials from the list **(4)**.



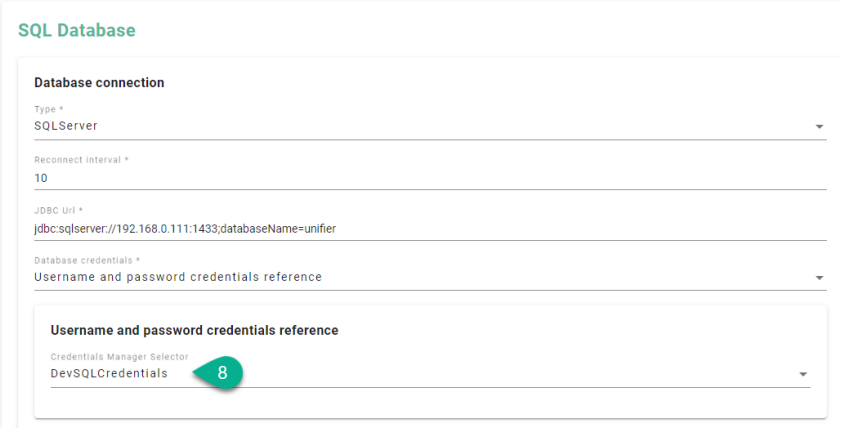
- If the credentials are not saved in the Credentials Manager, click on the **Add credentials** option **(5)**.



- Input the credentials details **(6)** and click on the **Save and Close** button **(7)**.



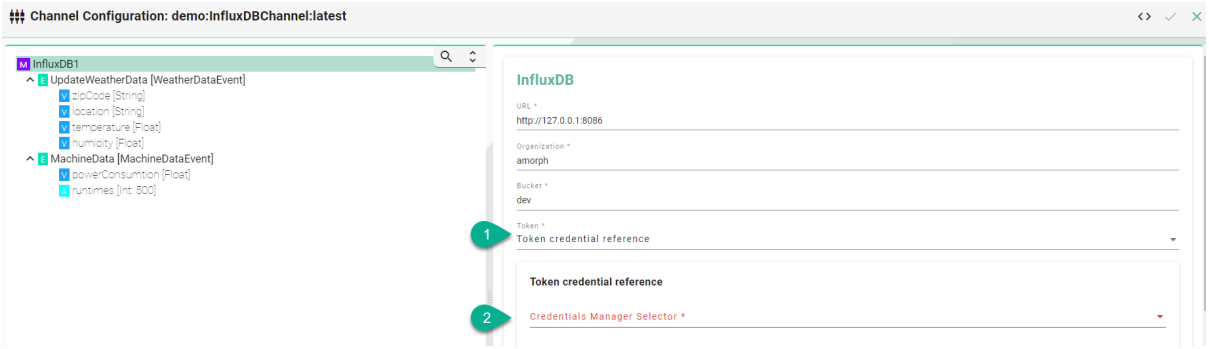
- The new credentials are saved and added into the configuration **(8)**.



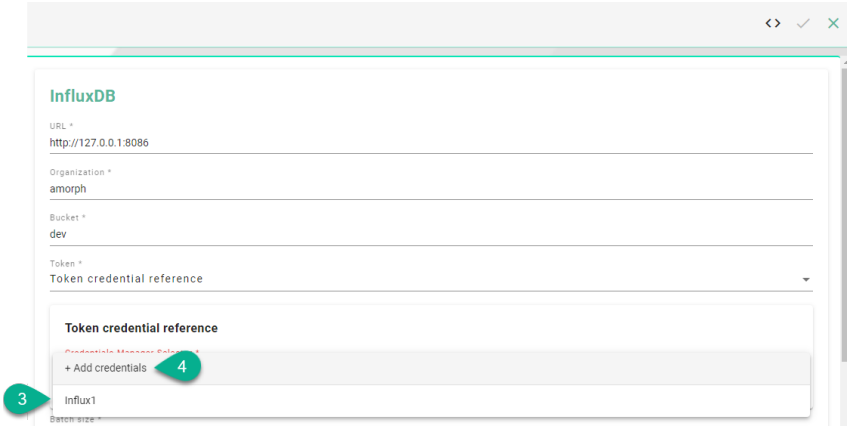
When configuring the Communication Channels, the user has the option to manually input a token or to select one from the Credential Manager.

Example of InfluxDB V2 Communication Channel configuration:

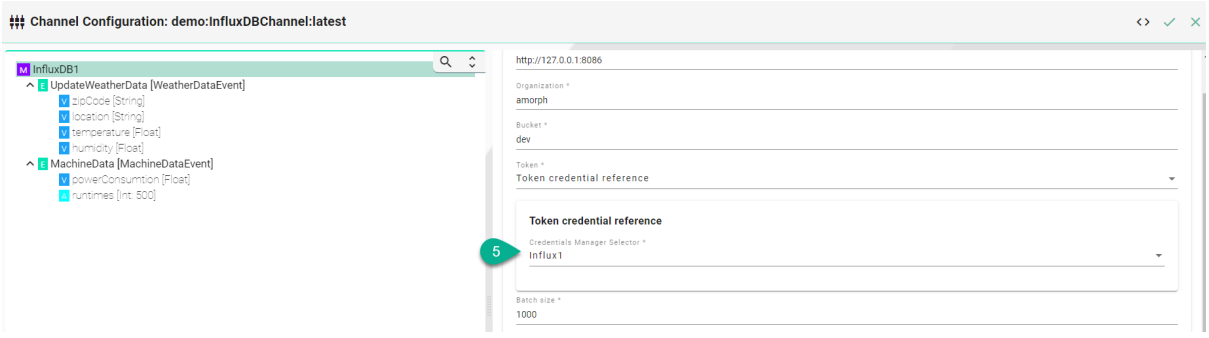
- For the **Token** field, select the **Token credentials reference** option **(1)**.



- Click on the **Credential Manager Selector** field **(2)**.
- Select a Token from the list **(3)**.



- If the token is not saved in the Credentials Manager, click on the **Add credentials** option **(4)**, input the token details and save it.
- The new token is saved and added into the configuration **(5)**.



Docker Java Image Manager

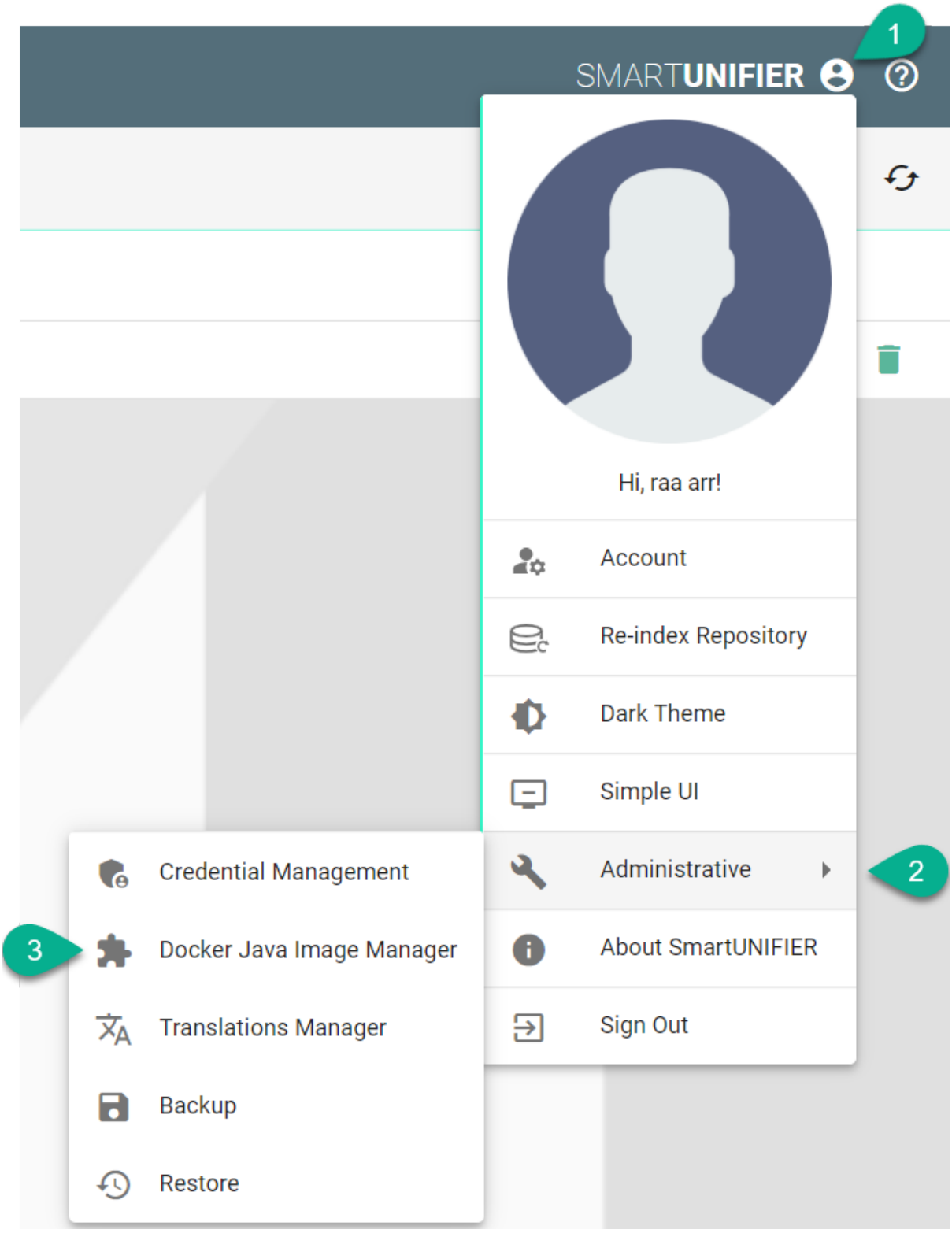
SMART**UNIFIER** supports the Deployment of Instances using Docker Containers using different Java base images. With the Docker Java Images Manager, the user can create and maintain different versions of Docker Java images.

This feature can only be accessed by a user with the administrator role.

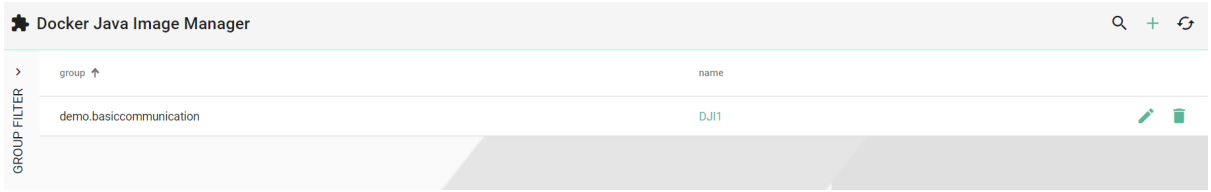
How to access

Follow the steps bellow to access the Docker Java Image Manager:

- Click on the **Account** icon **(1)**, go to **Administrative** section **(2)** and select the **Docker Java Image Manager** option **(3)**.



- The Docker Java Image Manager is visible.



Note

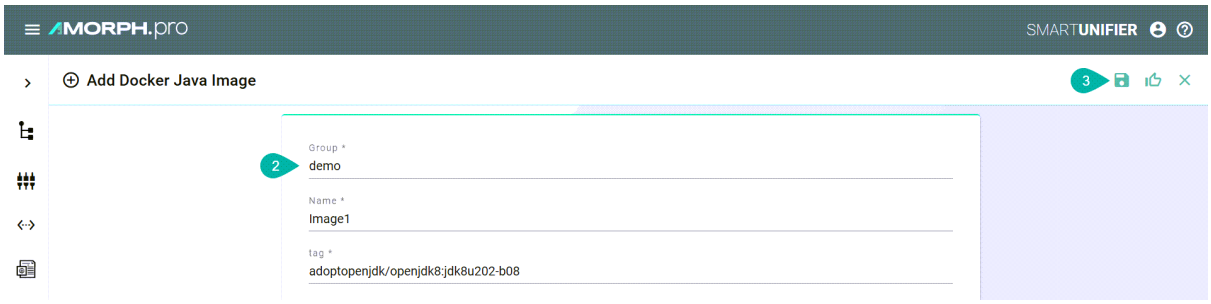
The Docker Java Image Manager can only be accessed by user accounts with an administrator role assigned.

Add a New Docker Java Image

Follow the steps described below to add a new Docker Java image:

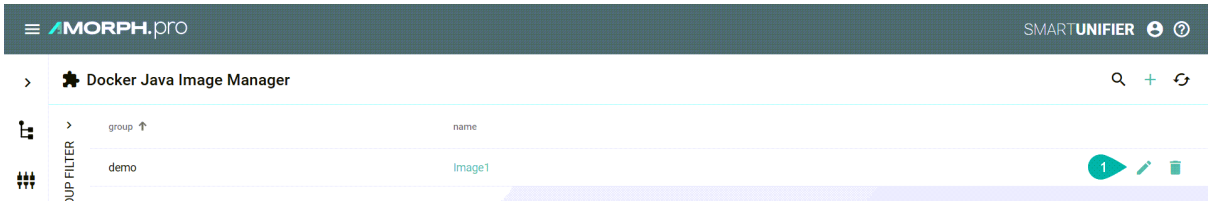
- Click on the **Add** button **(1)**.

- In the *Add Docker Java Image* view, a set of configuration parameters is required **(2)**: * Provide a **Group** and a **Name** * Provide a **tag** e.g., `adoptopenjdk/openjdk8:jdk8u202-b08`
- After all mandatory fields are filled in, click the **Save** button **(3)**.



Edit a Docker Java Image

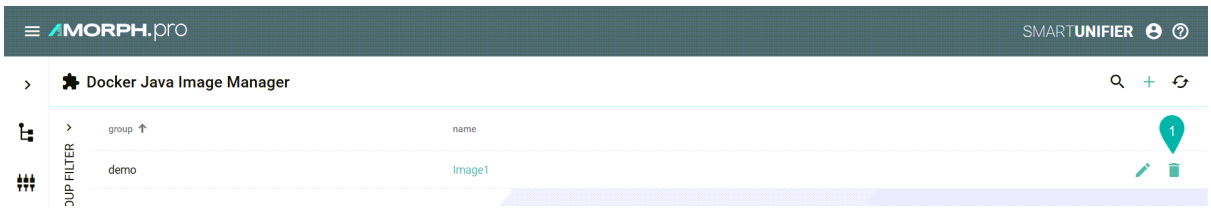
To edit a Docker Java image, select the **Edit** button **(1)**.



The Docker Java image is in the Edit Mode, the configuration parameters can be edited and then save the session by selecting the **Save** button.

Delete a Docker Java Image

To delete a Docker Java image, select the **Delete** button **(1)**.



A pop-up confirmation appears, select the **Delete** button.

Deployment Endpoints

What are Deployment Endpoints

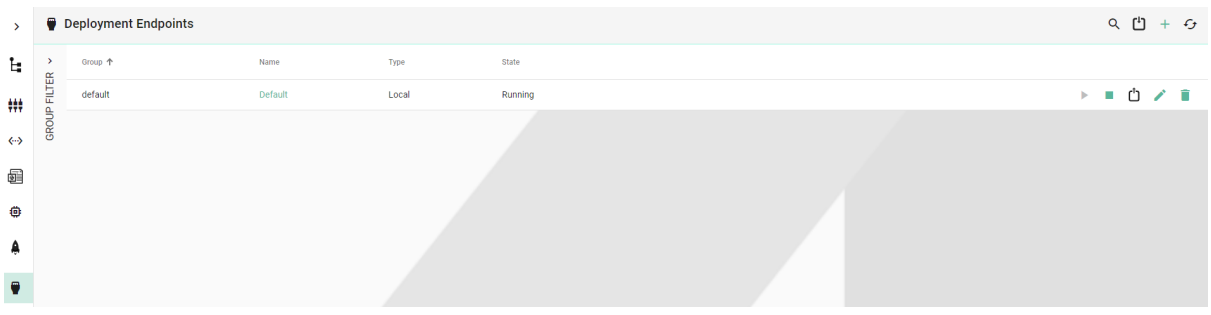
Deployment Endpoints are used to identify the location of a Deployment (i.e., the definition where an Instance is executed). With the Deployment Endpoints, you can create and maintain those locations. This feature can only be accessed by a user with the administrator role.

How to access

Follow the steps bellow to access the Deployment Endpoints:

- Click on the **Deployment Endpoints** button **(1)** to open the Deployment Endpoints perspective.

- The main view of the Deployment Endpoints is visible.



> **Note**
>
> The Deployment Endpoints can only be accessed by user accounts with an administrator role assigned.

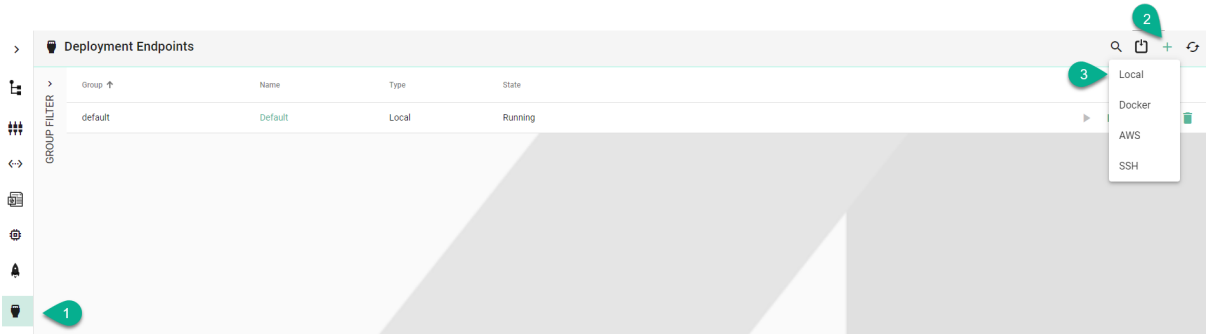
Deployment Endpoints Types

Local

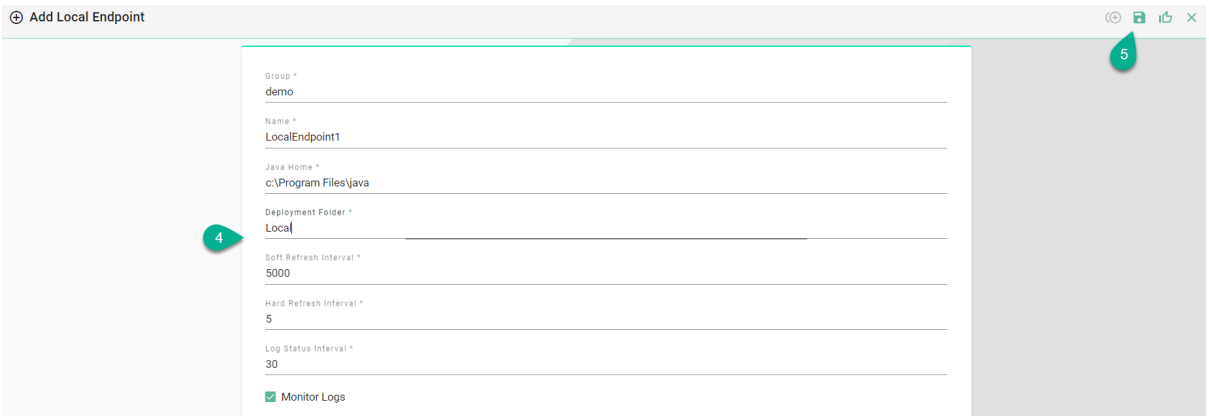
SMART**UNIFIER** supports Endpoint for Local Deployment. A **Default Local Endpoint** is preconfigured.

Follow the steps described below to create a Local Deployment Endpoint:

- Navigate to the SMART**UNIFIER** Deployment Endpoints perspective **(1)**.
- Click on the **Add Endpoint** button **(2)**.
- Select the Deployment Type **Local** from the pop-up **(3)**.



- In the **Add Endpoint** view a set of configuration parameters is required **(4)**
  - Provide a **Group** and a **Name**
  - Input the path for **Java**
  - Provide the **Deployment Folder**
  - Configure the **Soft/Hard Refresh Interval** and the **Log Status Interval** (in milliseconds)
  - Enable **Monitor Logs** (optionally)



- After all mandatory fields are filled in, click the **Save** button **(5)**.

Agent

SMART**UNIFIER** allows Communication Instances to be deployed on any machine with an active Agent process. The Agent enables communication between the SMART**UNIFIER** Manager and the Communication Instance.

Installation as a Windows Service

Follow the steps below to install and operate SMARTUNIFIER Agent as a Service under Windows:

- Move the SMART**UNIFIER** Agent package to a suitable location
- Extract the **.zip**-archive
- Open a terminal window with *Administrator privileges* within the package
- Execute the following commands in the terminal window to:

Listing 1: Install

```
SmartUnifierAgentService.bat install
```

Listing 2: Start

```
SmartUnifierAgentService.bat start
```

Listing 3: Stop

```
SmartUnifierAgentService.bat stop
```

Listing 4: Uninstall

```
SmartUnifierAgentService.bat uninstall
```

Installation as a Java Process

Follow the steps below to install and operate SMART**UNIFIER** Agent as a Process under Windows:

- Move the SMART**UNIFIER** Agent package to a suitable location
- Extract the **.zip**-archive
- Execute the **SmartUnifierAgent.bat** script

Note

The console is for information purposes only. It can be moved to any suitable location on your screen or it can be hidden. Nevertheless, do not close it, because the related processes will also be terminated.

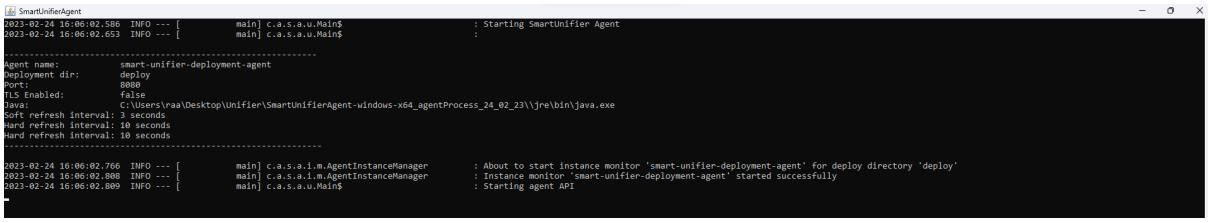
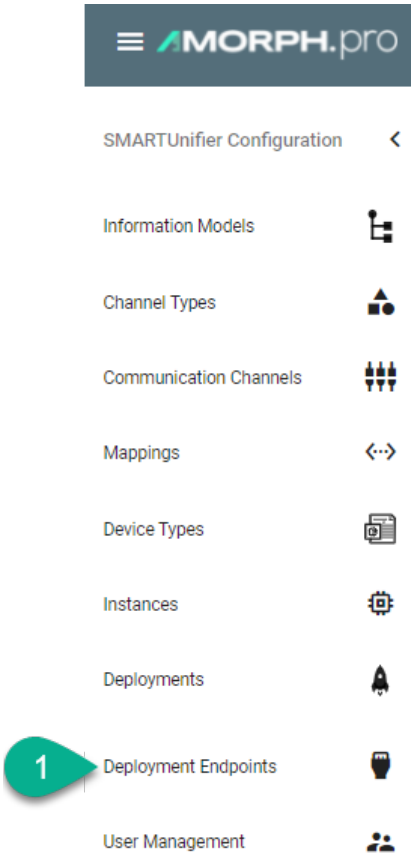


Fig. 1: Example of an Agent running on a remote machine

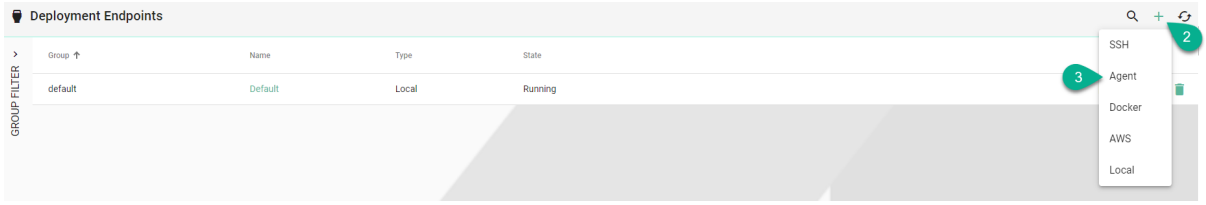
Creating a Deployment Endpoint

After installing the agent, you need to register the endpoint with the SMART**UNIFIER** Manager. Follow the steps below to create the deployment endpoint:

- Navigate to the SMART**UNIFIER** Deployment Endpoints perspective **(1)**.



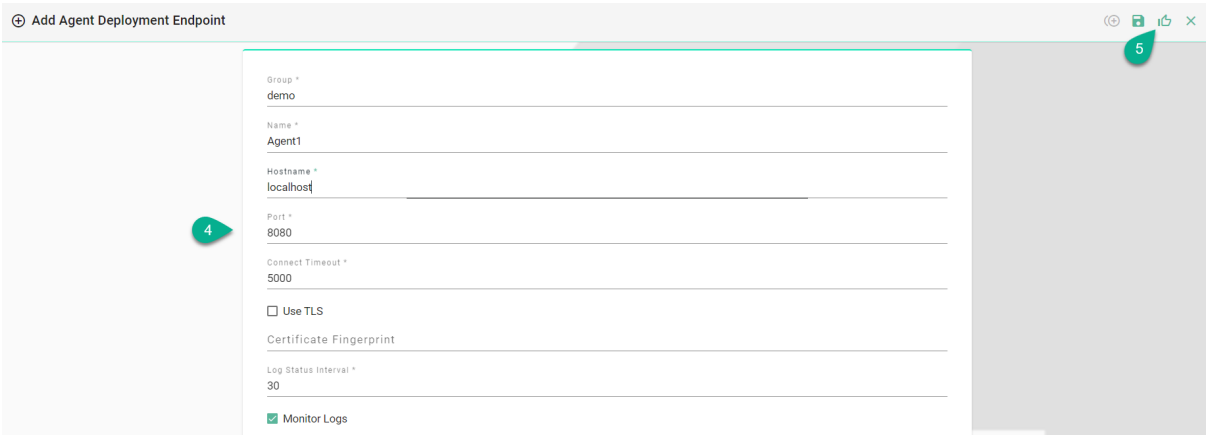
- Click on the **Add Endpoint** button **(2)**.
- Select the Deployment Type **Agent** from the pop-up **(3)**.



- In the **Add Endpoint** view a set of configuration parameters is required **(4)**
  - Provide a **Group** and a **Name**

- Provide the VM **Hostname** (Default port is: **8080**)
- Set the **Connection Timeout**
- If needed, check the **useTls** box and input certificates for secured connections
- Set the **Log Status Interval**
- Enable the **Monitor Logs**

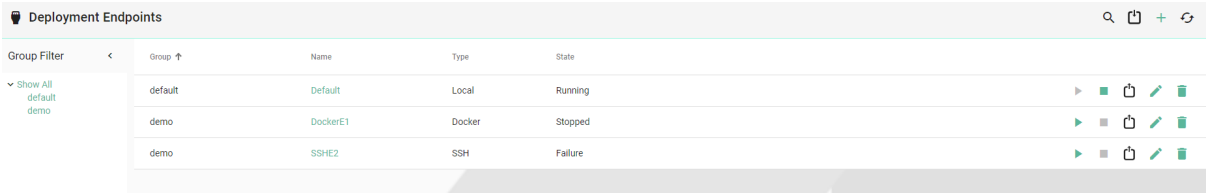
- After all mandatory fields are filled in, click the **Save and Close** button **(5)**.



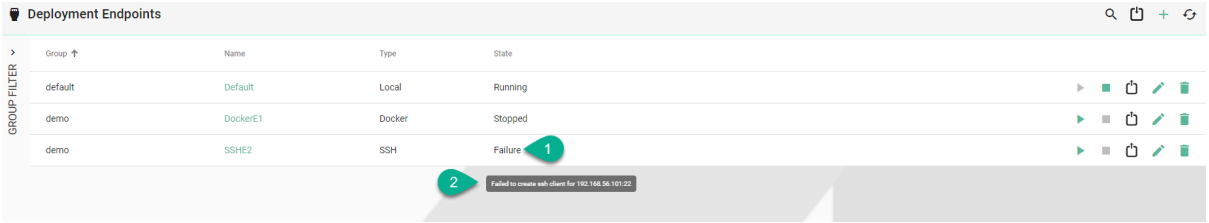
Deployment Endpoints States

A Deployment Endpoint can have the following states:

- **Stopped** - The Stop command has been sent and the Deployment Endpoint is stopped
- **Starting** - The Start command has been sent
- **Running** - Deployment Endpoint is up and running
- **Failure** - The Start command has been sent and the Deployment Endpoint has failed to start



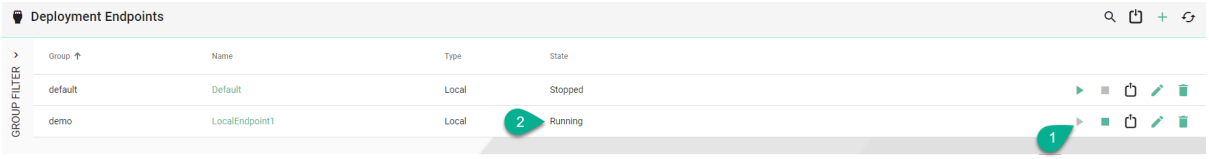
For the **Failure** state, hover over it **(1)** and a pop-up will display the error **(2)**.



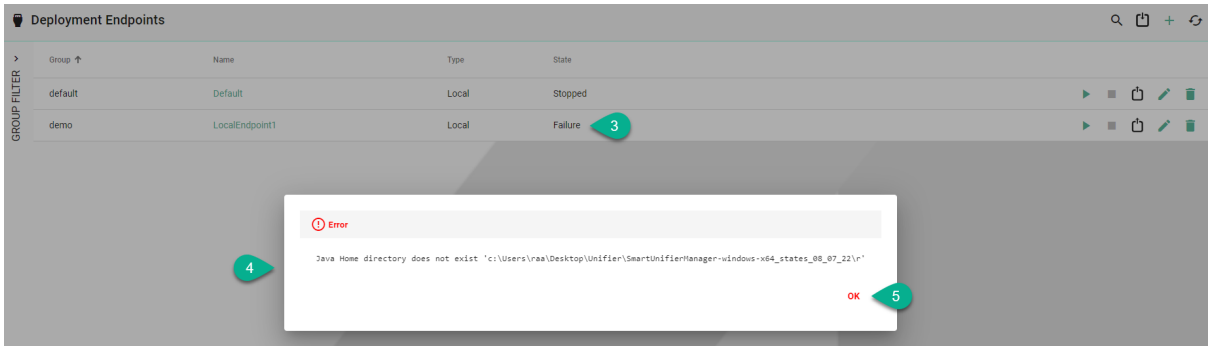
Deployment Endpoints Operations

Start Endpoint

After a Deployment Endpoint is created, its default state is Stopped. To start it, click on the **Start** button **(1)**. The state will change into **Starting** and if it succeeds, the state becomes **Running (2)**.



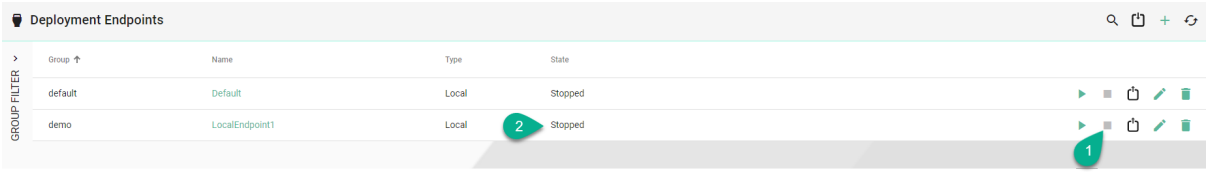
If the Deployment Endpoint fails to start, the state changes into **Failure (3)** and an error message will be displayed **(4)**.



Click on the **OK** button **(5)** to close the error message.

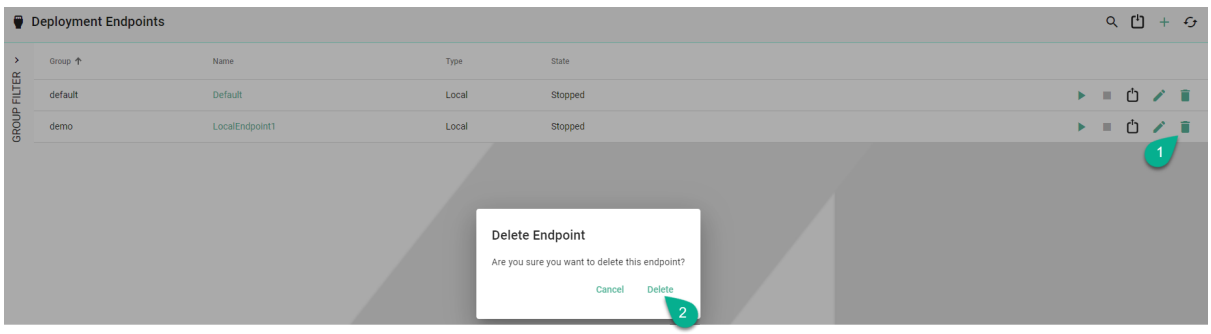
Stop Endpoint

To stop a Deployment Endpoint, click on the **Stop** button **(1)** and the state will change accordingly **(2)**.



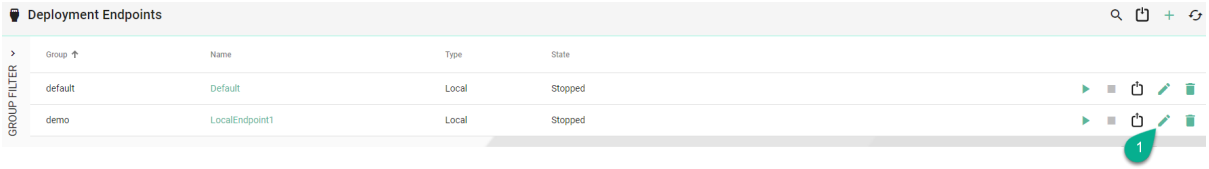
Delete Endpoint

To remove a Deployment Endpoint, click on the **Delete** button **(1)** and confirm the action **(2)**.

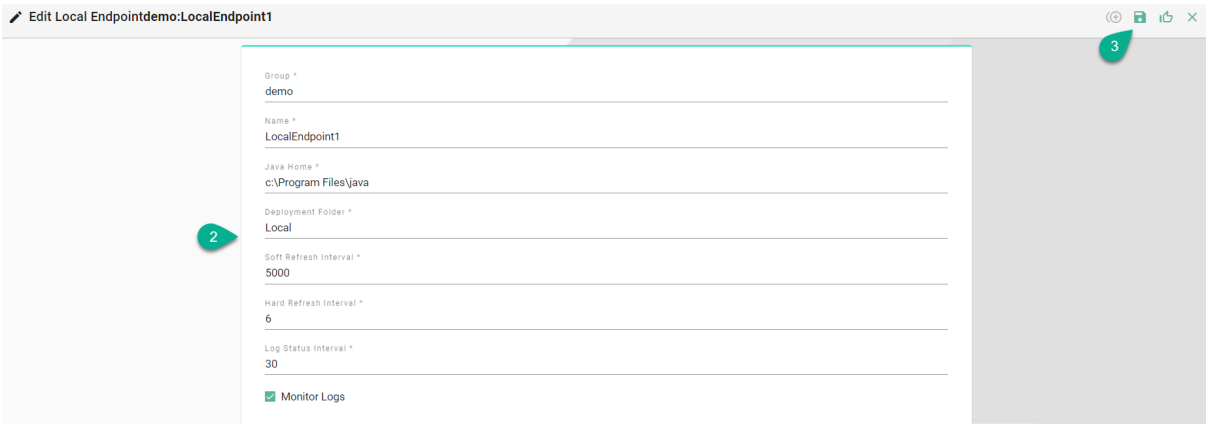


Edit Endpoint

To edit a Deployment Endpoint, click on the **Edit** button **(1)**.



In the Deployment Endpoint edit view update the configuration **(2)** and click on the **Save** button **(3)**.

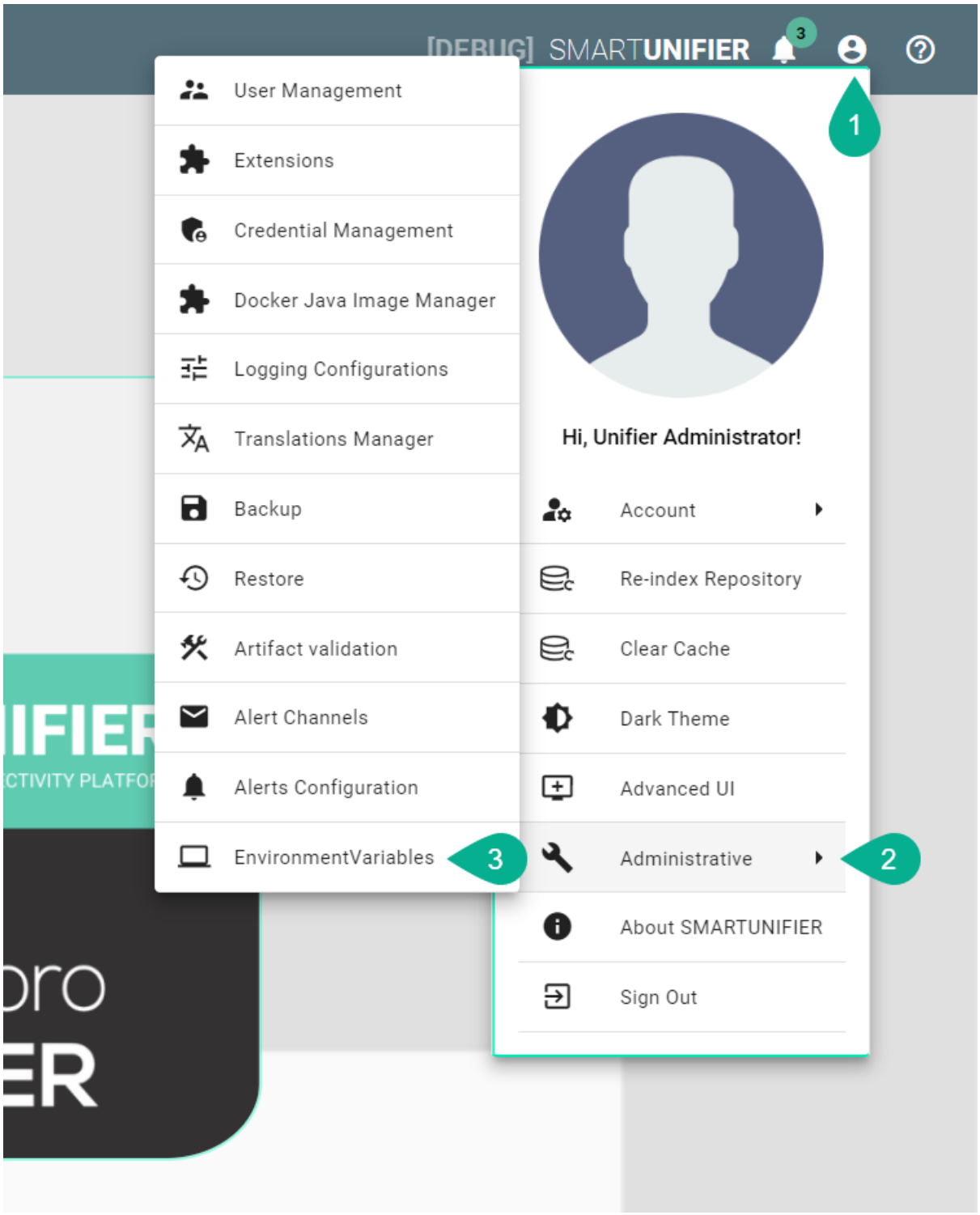


Environment Variables

Environment Variables can be used within the Channel configuration to store values that can be used across multiple Channels. This allows you to define a value once and use it in multiple places, making it easier to manage and update values across multiple Channels.

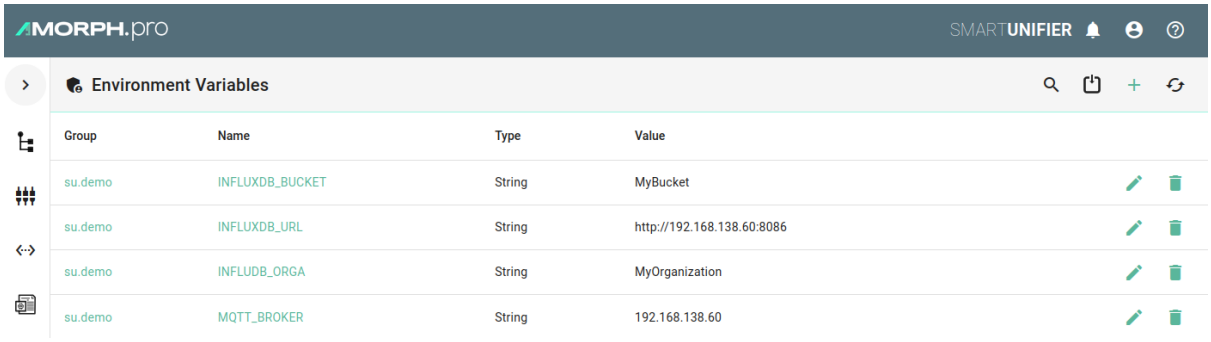
How to access

1. Click on the **Account icon** on the top right corner of the screen.
2. Select **Administrative**
3. Select **Environment Variables**

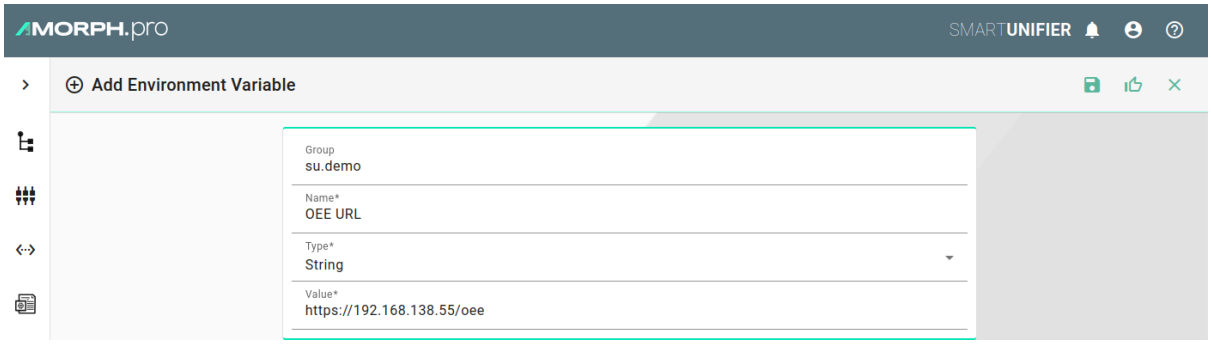


Managing Environment Variables

In the following menu, Environment variables can be added, modified and deleted



Environment Variable Properties

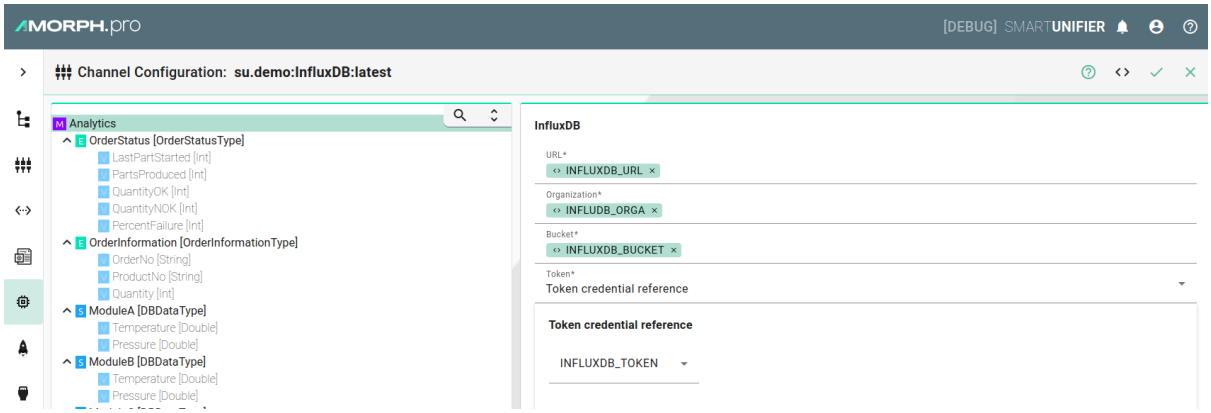


Property	Description	Example
Group	The group to which the environment variable belongs.	su_demo
Name	The name of the environment variable.	OEE_URL
Type	The data type of the environment variable.	String
Value	The value assigned to the environment variable.	https://192.168.138.55/oee

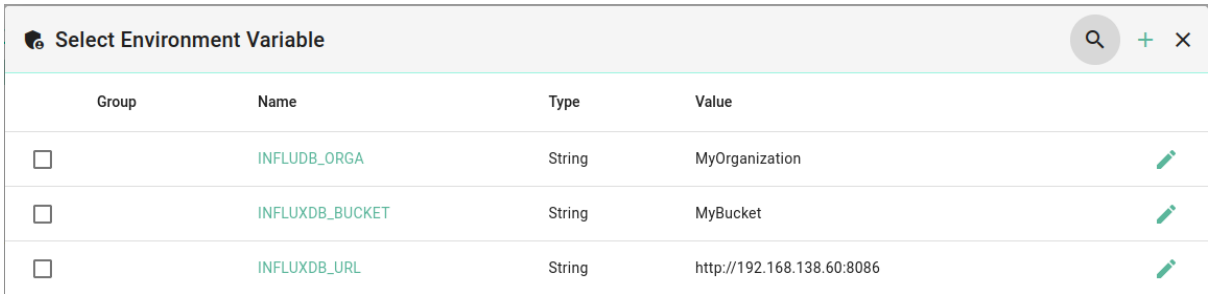
Using Environment Variables

To use an Environment Variable in the Channel configuration, use the shortcut CTRL + SHIFT + E. In the popup that appears, you can select the desired environment variable by clicking the checkbox on the left.

Example channel configuration with Environment Variables

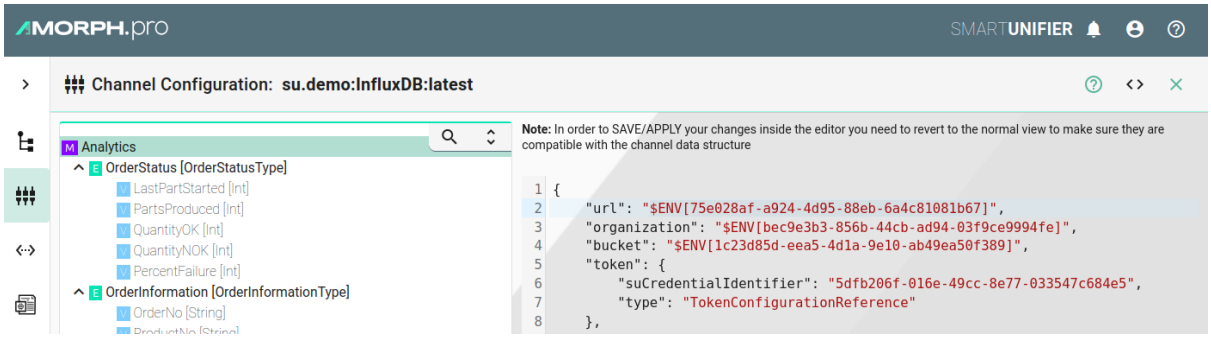


Pupup for selecting / adding Environment Variables



Inside SmartUnifier

Environment Variables are stored inside the internal database, and each Environment Variable has its own UUID. When an Environment Variable is used within a channel configuration, a placeholder will be inserted (e.g., `$ENV[75e028af-a924-4d95-88eb-6a4c81081b67]`). When saving, the placeholder will be temporarily replaced by the actual variable to perform validation of the configuration. During deployment, the placeholder will then be replaced by the actual value stored in the Environment Variable database.



Extensions

> **Note**
>
> Please contact Amorph Systems for guidance on how to enable and use extensions.

How to install extensions

Request the specific plugin from Amorph Systems. Follow the steps below to install it:

1. Navigate to **..SmartUnifierManager/plugins**
2. **Create a new directory with the name of the plugin in this format:**
    - SmartUnifierPluginJsonToModel
    - SmartUnifierPluginAwsSiteWise or
    - SmartUnifierPluginOpcUaToModel
3. Add the jar into the specified directory
4. Restart the SMART**UNIFIER** Manager

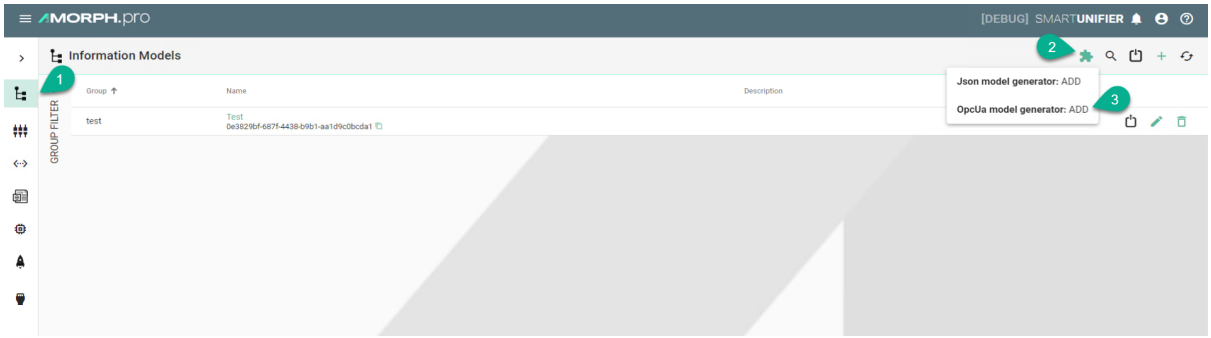
OpcUa Model Import

SMART**UNIFIER** provides the possibility to generate an OpcUa Information Model using a XML-file or connecting to the OpcUa server.

Create a new Information Model (OPCUA)

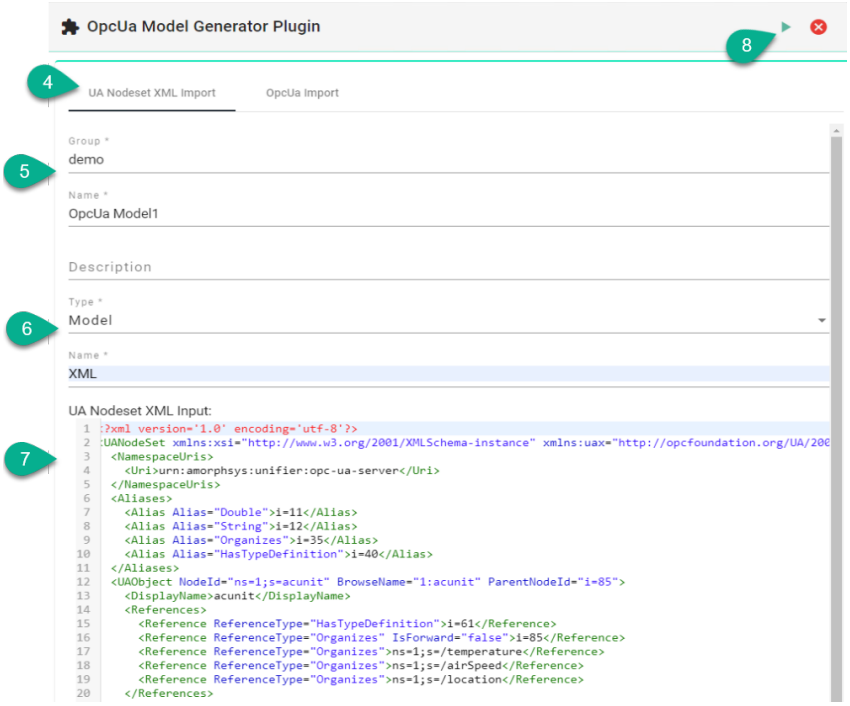
Follow the steps described below to generate an Information Model:

- Select the SMART**UNIFIER** Information Model Perspective **(1)**.
- Click on the **Extensions** button **(2)**.
- Select the **OpcUa model generator: ADD** option **(3)**.

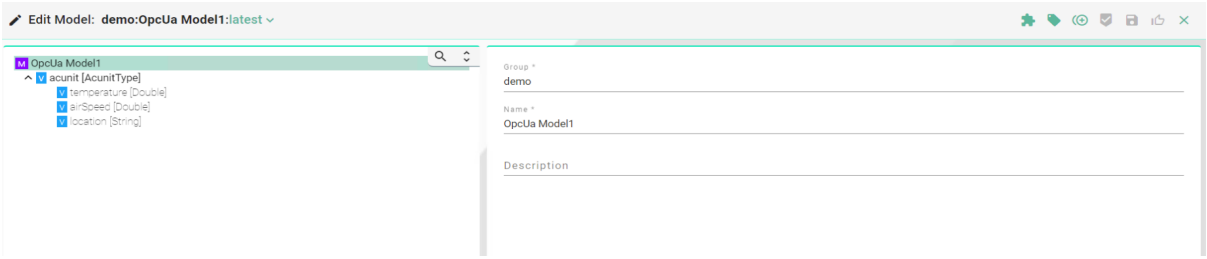


OpcUa Nodeset XML Import

- Select the **UA Nodeset XML Import** option **(4)**.
- Provide the following mandatory information: **Group** and **Name (5)**.
- Select the type of the **Information Model Node** and provide a **Name (6)** :
    - **Model** - the OpcUa data is converted inside the root model node
    - **Event** - the OpcUa data is converted inside an Event node type
    - **Variable** - the OpcUa data is converted inside a Variable node type
- Paste the content from the XML file **(7)**.
- To finish, click on the **Save** Button **(8)**.

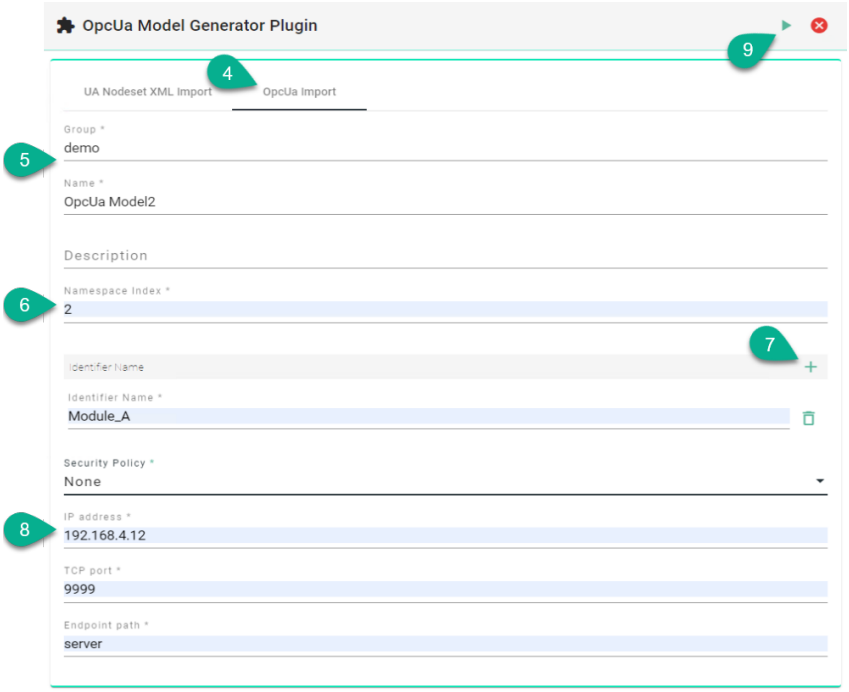


- The Information Model is generated.

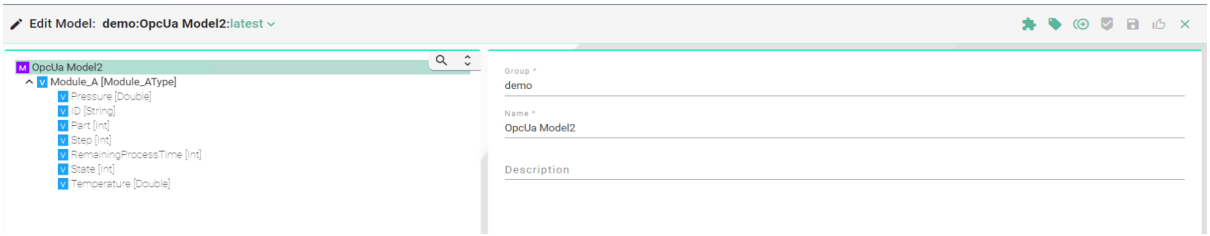


OpcUa Direct Import

- Select the **OpcUa Import** option **(4)**.
- Provide the following mandatory information: **Group** and **Name (5)**.
- Input the **Namespace Index (6)**.
- Click on the **Add identifier name** button and provide an **Identifier Name (7)**.
- Provide the server details **(8)**:
  - **Security Policy**
  - **IP address**
  - **TCP port**
  - **Endpoint path**
- To finish, click on the **Save** Button **(9)**.



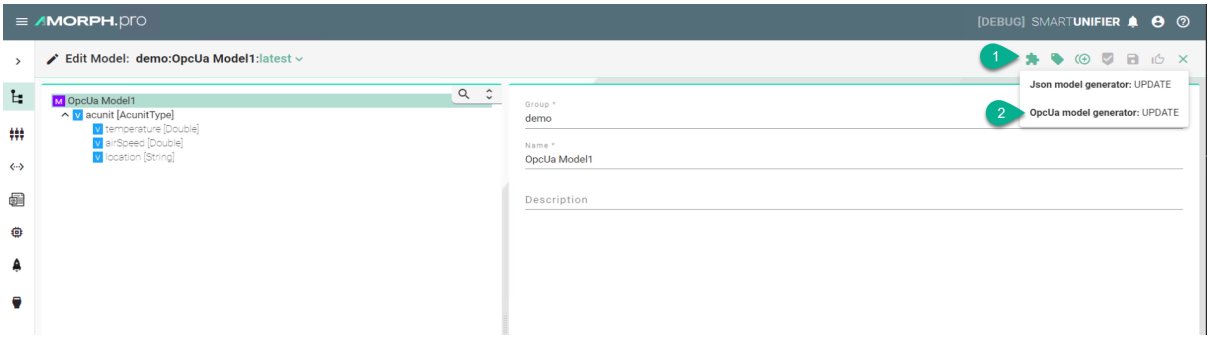
- The Information Model is generated.



Update an existing Information Model (OPCUA)

Follow the steps described below to update an Information Model:

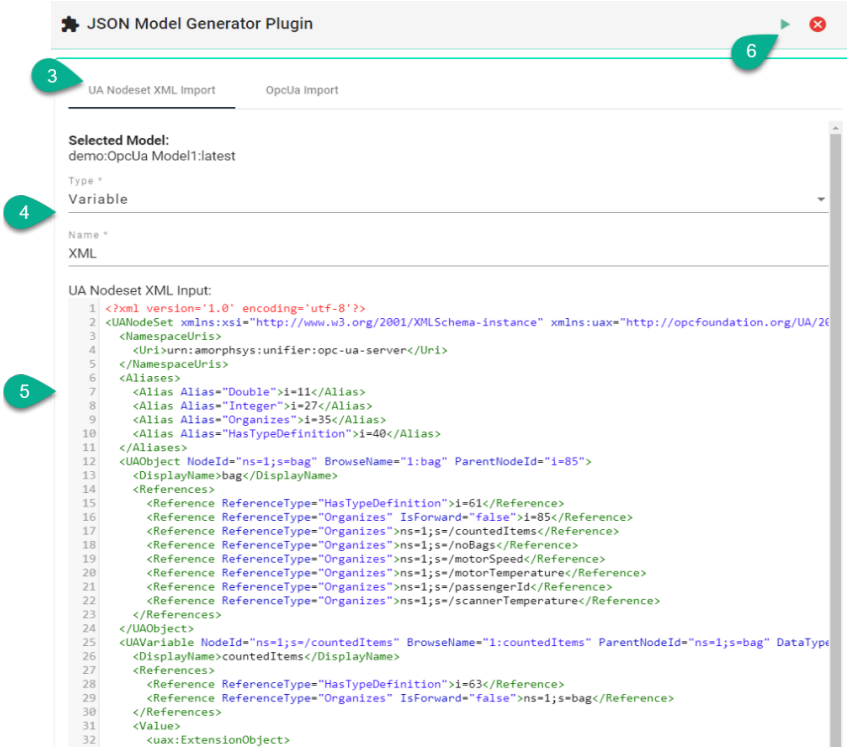
- Open an Information Model to edit and click on the **Extensions** button **(1)**.
- Select the **OpcUa model generator: UPDATE** option **(2)**.



OpcUa Nodeset XML Import (Update)

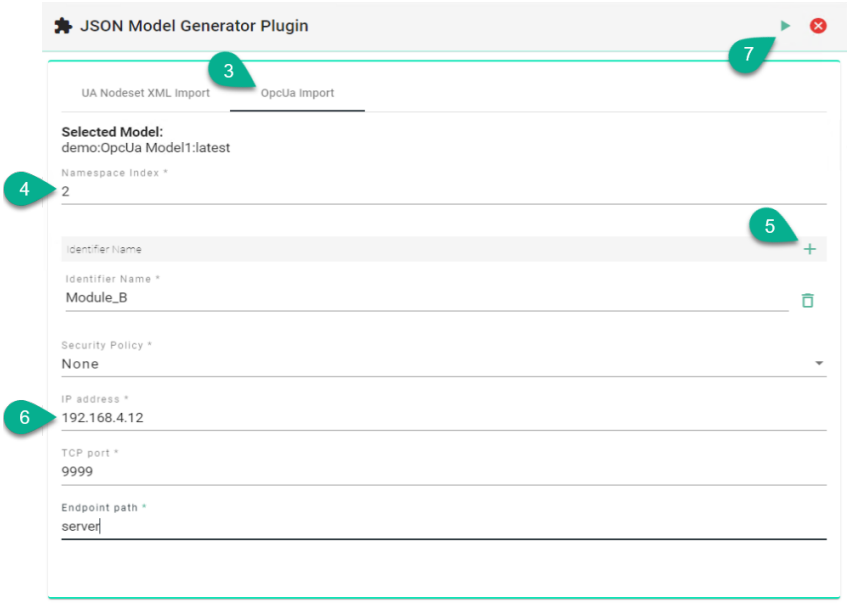
- Select the **UA Nodeset XML Import** option **(3)**.
- Update the **Type** and the **Name (4)**.
- Paste the updated content from a XML-file **(5)**.

- To finish, click on the **Save** button **(6)**.



OpcUa Direct Import (Update)

- Select the **OpcUa Import** option **(3)**.
- Input the **Namespace Index (4)**.
- Click on the **Add identifier name** button and provide an **Identifier Name (5)**.
- Provide the server details **(6)**:
  - **Security Policy**
  - **IP address**
  - **TCP port**
  - **Endpoint path**
- To finish, click on the **Save** Button **(7)**.



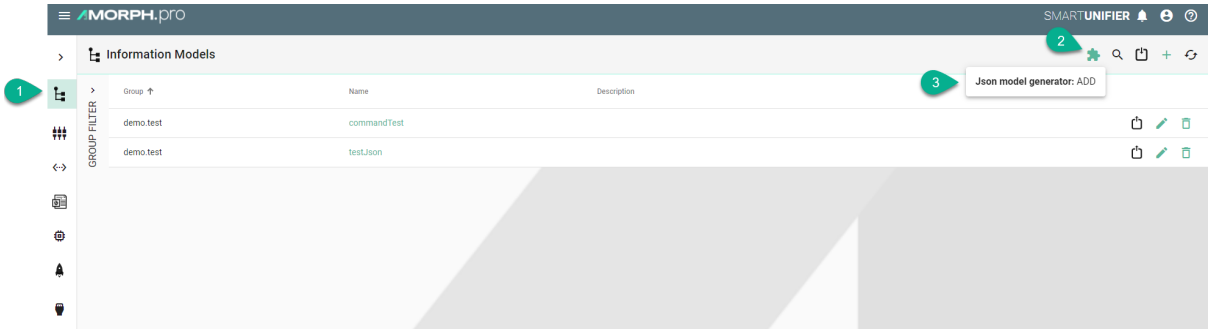
JSON Model Import

SMART**UNIFIER** provides the possibility to generate an Information Model using a JSON-file.

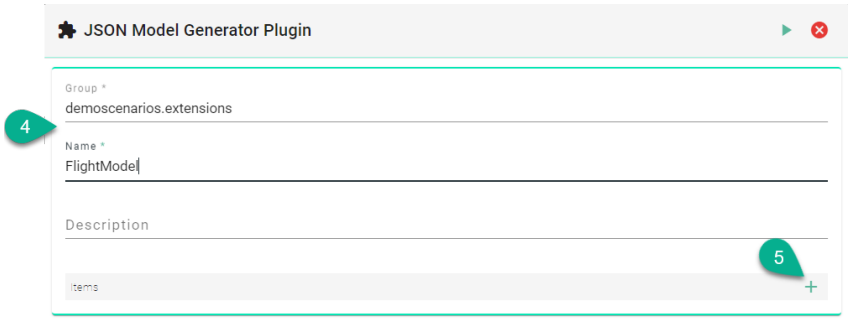
Create a new Information Model (JSON)

Follow the steps described below to generate an Information Model:

- Select the SMART**UNIFIER** Information Model Perspective **(1)**.
- Click on the **Extensions** button **(2)**.
- Select the **Json model generator: ADD** option **(3)**.



- Provide the following mandatory information: **Group** and **Name (4)**.
- Click on the **Add item** button **(5)**.

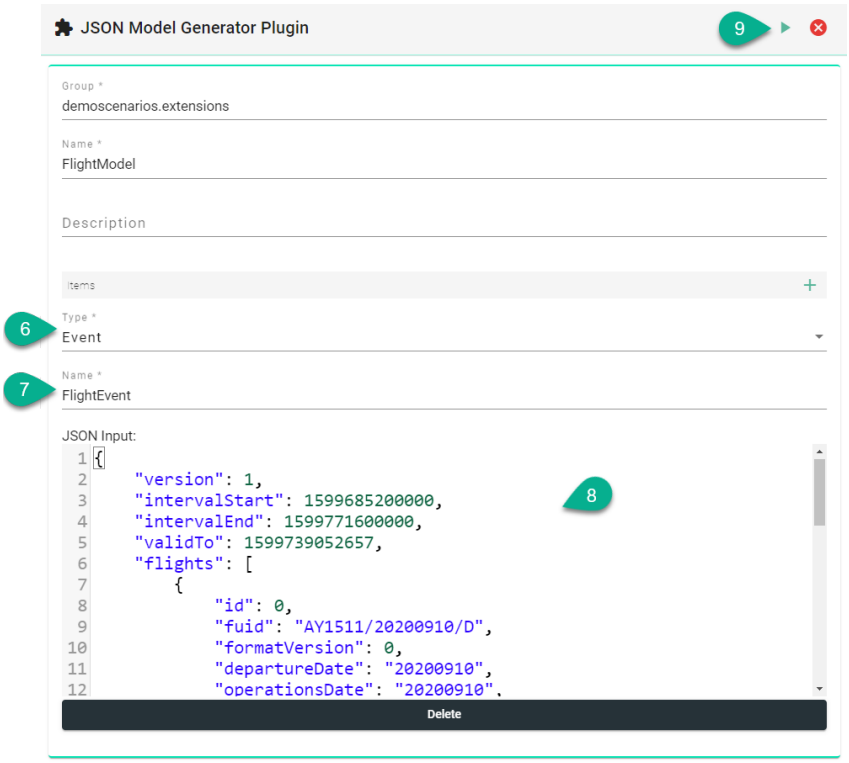


- Select the type of the **Information Model Node (6)**:
  - **Model** - the Json data is converted inside the root model node
  - **Event** - the Json data is converted inside an Event node type
  - **Variable** - the Json data is converted inside a Variable node type
  - **Command** - the Json data is converted inside a Command node type
- Enter a **Name (7)**.
- Paste the content from a Json file **(8)**.

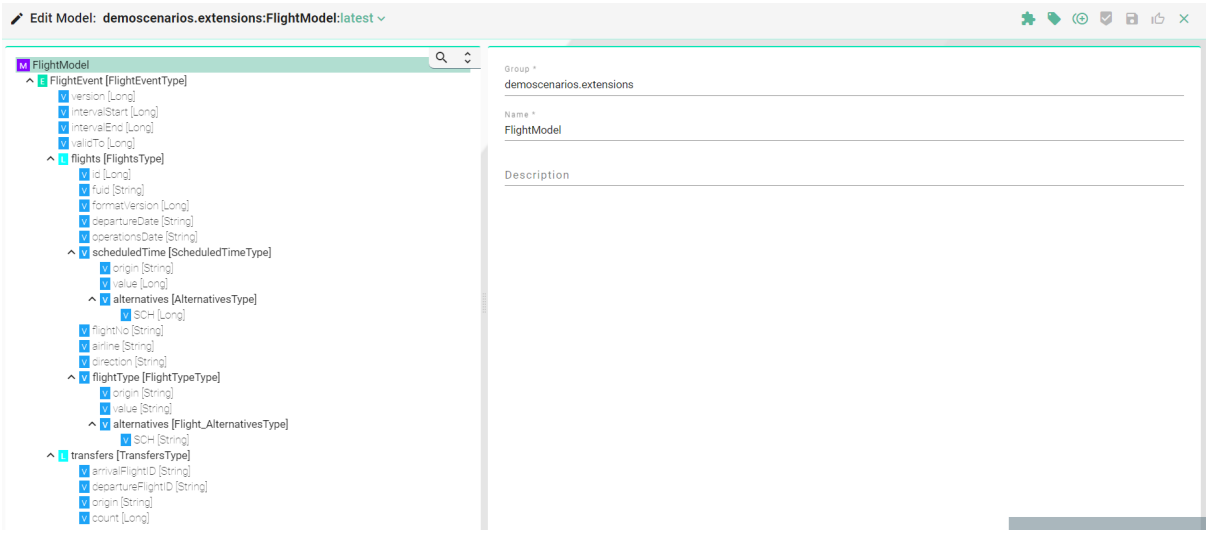
Note

Make sure to copy the JSON object { }.

- To finish, click on the **Save** Button **(9)**.



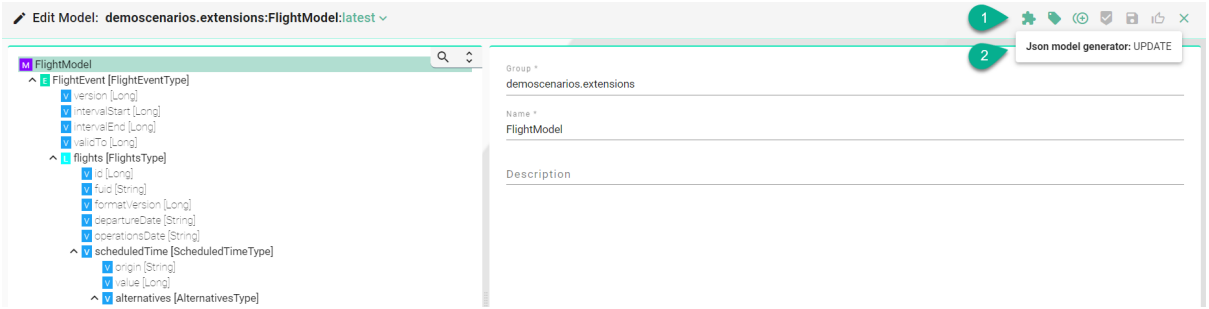
- The Information Model is generated.



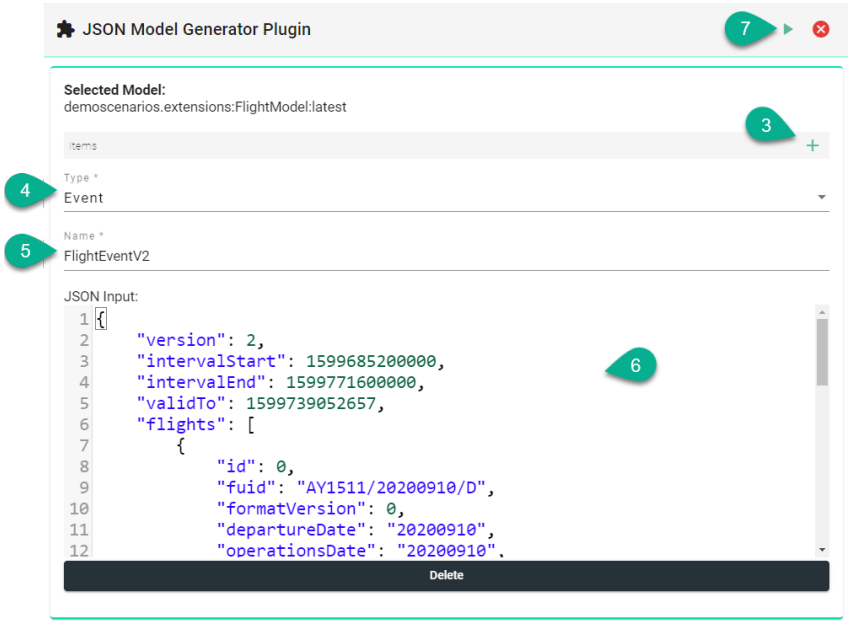
Update an existing Information Model (JSON)

Follow the steps described below to update an Information Model:

- Open an Information Model to edit and click on the **Extensions** button **(1)**.
- Select the **Json model generator: UPDATE** option **(2)**.



- Click on the **Add item** button **(3)**.
- Update the **Type (4)** and the **Name (5)**.
- Paste the updated content from a Json-file **(6)**.
- To finish, click on the **Save** button **(7)**.



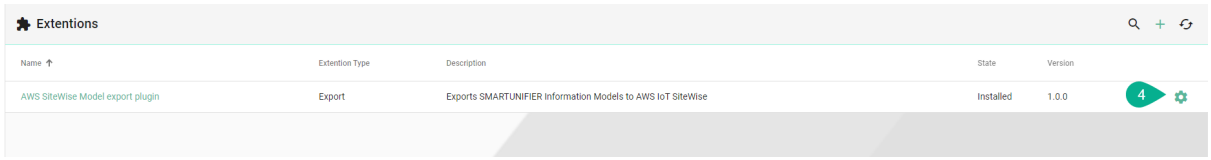
AWS IoT SiteWise Model Export

This extension allows you to export an SMART**UNIFIER** Information Model to AWS IoT SiteWise.

How to access

To access the AWS IoT SiteWise extension, click on the Account icon (1), go to the Administrative option (2) and select the Extensions (3).

Then select the **configuration** button of the (4)



How to export Information to AWS IoT SiteWise

Prerequisite

We recommend to have one user dedicated for SMART**UNIFIER**.

Attach the following permission:

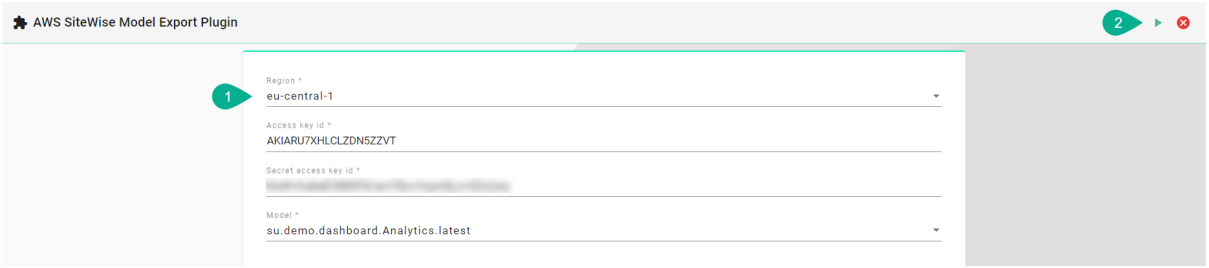
Policy ARN	Description
arn:aws:iam::aws:policy/AWSIoTSiteWiseFullAccess	Provides full access to IoT SiteWise.

If you do not have already an access key available you have to create a new access key. We recommend to create a new access key after 90 days.

Configuration

Follow the steps described below to export a the SMART**UNIFIER** Information Model:

- Configuration of the extension (1):
  - Select the region of the AWS Iot SiteWise service you are using
  - Enter the **access key id** and the **secret access key id**
  - Select the **Information Model** you want to export
- Click on the **Run** button to execute the export (2)



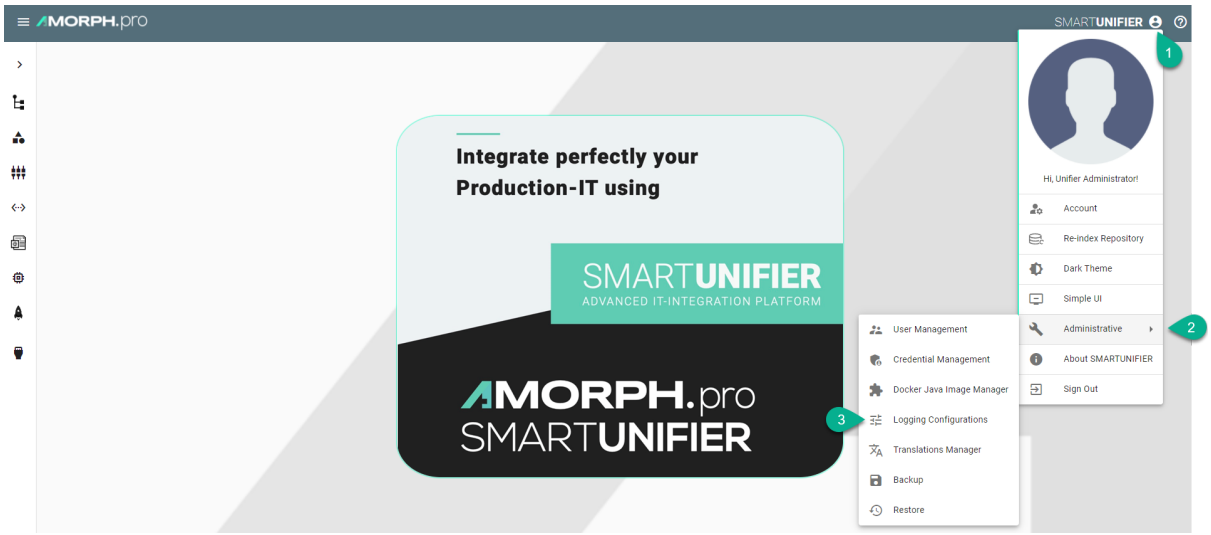
Logging Configurations

Log files within SMART**UNIFIER** are created through the Logback logging framework. The Logging Configuration functionality allows for the creation of new *log level* configurations. The configuration can be used during the deployment of a Communication Instance.

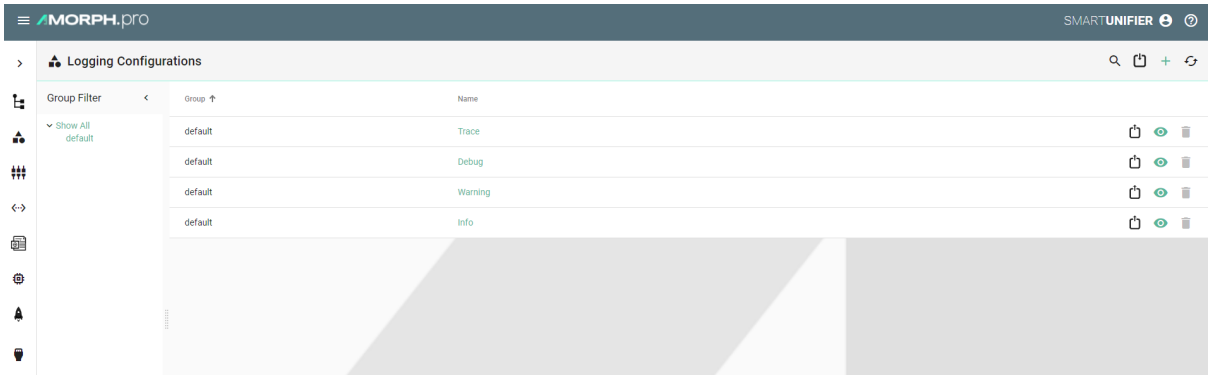
Accessing the Feature

To access the feature, follow the steps below:

- Click on the **Account** icon **(1)**
- Navigate to the **Administrative** option **(2)**
- Select the **Logging Configurations** perspective **(3)**



When you reach the Logging Configurations main view, you will see it as depicted below. There are four predefined log level configurations available, which can serve as templates for creating a new log level.



Note

This feature can be only used by users with the administration role.

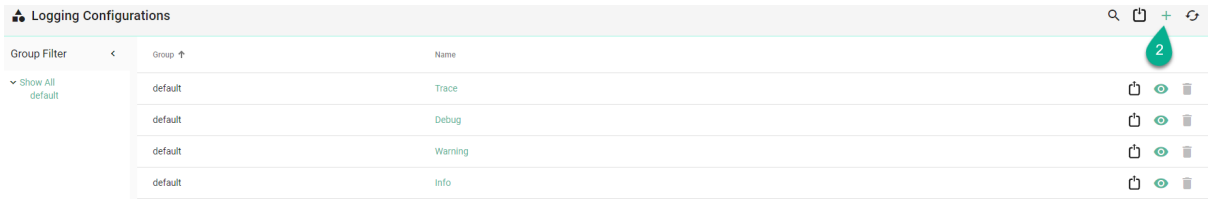
Note

The predefined logging configurations cannot be edited or deleted.

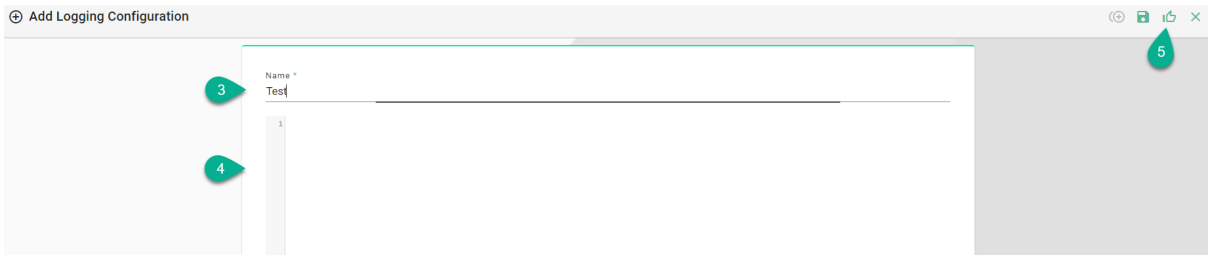
Add a New Logging Configuration

To add a new logging configuration, follow the steps below:

- Navigate to the **Logging Configurations** perspective
- Click on the "Add" button **(2)**



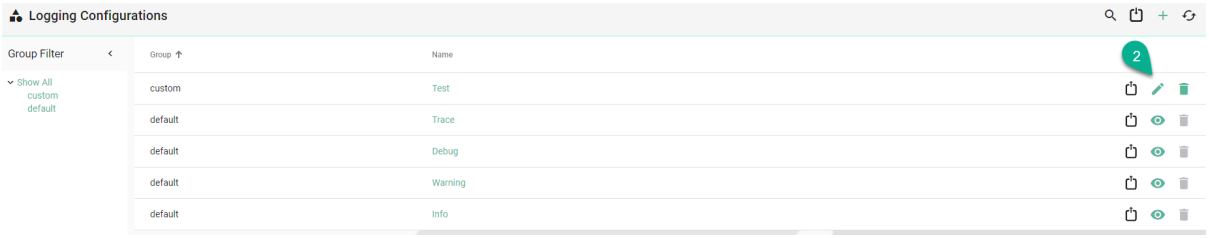
- Enter the file **Name (3)** and the configuration details **(4)**
- Click on the **Save and Close** button to finish **(5)**



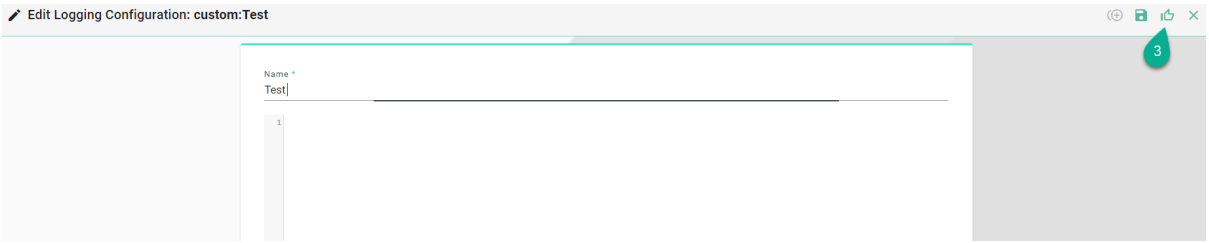
Edit a Logging Configuration

To edit a logging configuration, follow the steps below:

- Navigate to the **Logging Configurations** perspective
- Click on the "Edit" button **(2)**



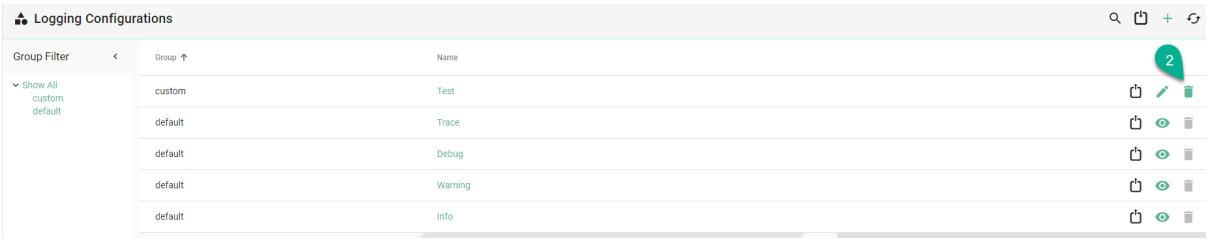
- Edit and click on the **Save and Close** button to finish **(3)**



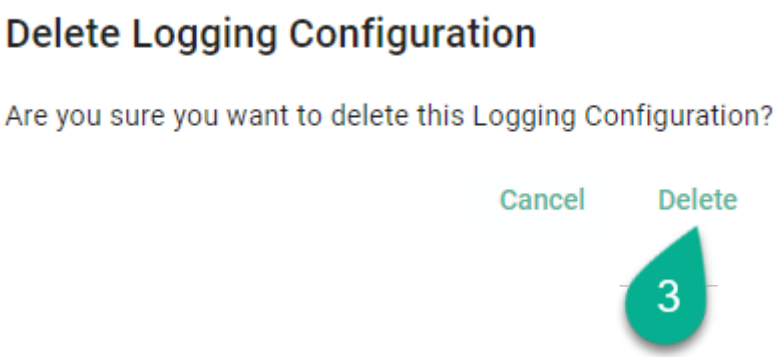
Delete a Logging Configuration

To delete a logging configuration, follow the steps below:

- Navigate to the **Logging Configurations** perspective
- Click on the "Delete" button **(2)**



- To confirm, click on the **Delete** button **(3)**



User Management

About User Management

Within the User Management the administrator can create users accounts, assign permissions as well as activate or deactivate user accounts.

User Roles and Permissions

This table outlines the permissions and capabilities available to different roles within SMART**UNIFIER**, segmented into general features and tasks.

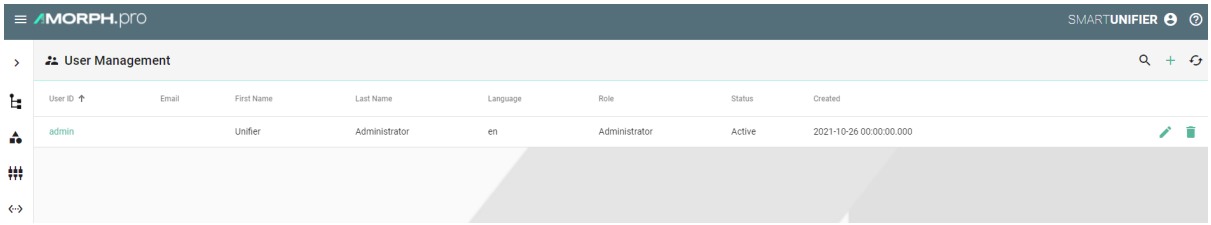
Feature	Reader	Writer	Administrator
View Information Models	X	X	X
Add/Edit Information Models	-	X	X
View Communication Channels	X	X	X
Add/Edit Communication Channels	-	X	X
View Mappings	X	X	X
Add/Edit Mappings	-	X	X
View Device Types	X	X	X
Add/Edit Device Types	-	X	X
View Communication Instances	X	X	X
Add/Edit Communication Instances	-	X	X
View Deployments	X	X	X
Add/Edit Deployments	-	X	X
Add/Edit Deployment Endpoints	-	X	X
Clear Cache	-	X	X
Add/Edit Channel Types (Advanced UI)	-	X	X
User Management	-	-	X
Extensions	-	-	X
Credential Management	-	-	X
Docker Java Image Manager	-	-	X
Logging Configurations	-	-	X
Translation Manager (Debug Mode)	-	-	X
Backup	-	-	X
Restore	-	-	X
Artifact Validation	-	-	X
Re-index Repository	-	-	X
Alert Channels	-	-	X
Alert Configuration	-	-	X
Environmental Variables	-	-	X

How to access

Follow the steps bellow to access the User Management:

- Click on the **Account** icon **(1)**, go to the **Administrative** option **(2)** and select the **User Management** perspective **(3)**.

- The User Management main view is visible.



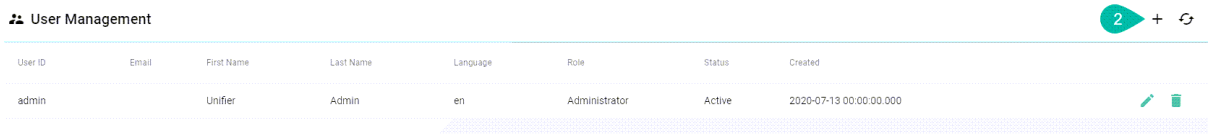
> **Note**
>
> The User Management can only be accessed by user accounts with an administrator role assigned.

Add a new user

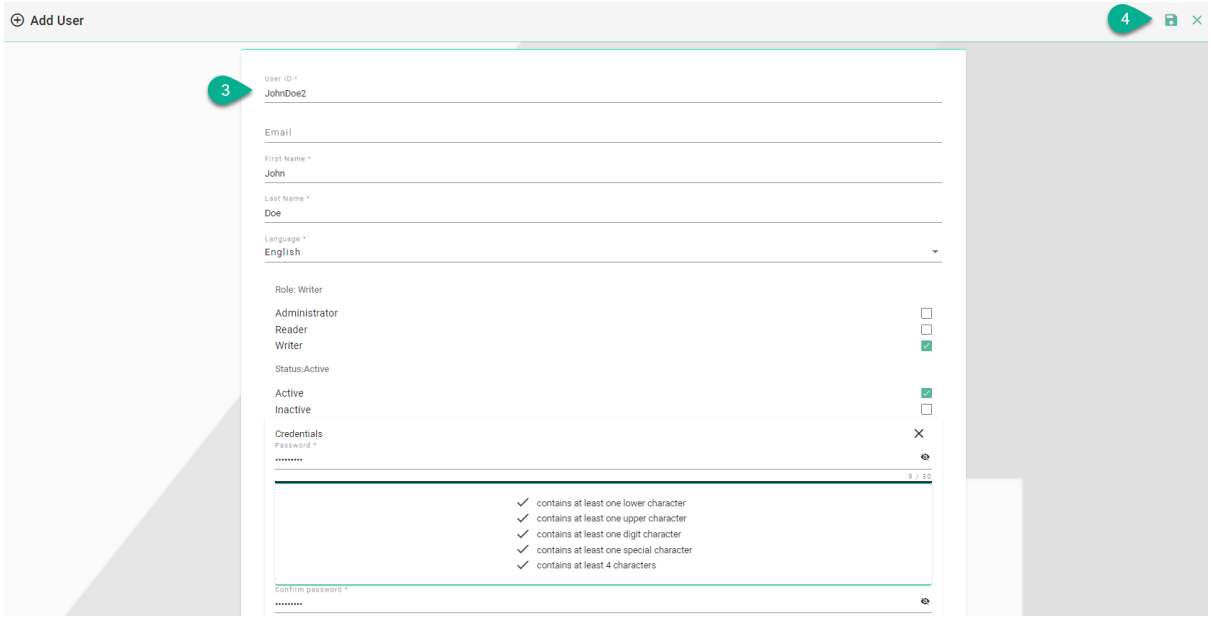
This procedure describes how to create a new user account.

- Select the SMART**UNIFIER** User Management perspective **(1)**.

- Click the "Add User" button **(2)**.



- In the "Add User" view provide the following information **(3)**:
  - Provide a **user id**, **first** and **last name**
  - Optionally, provide an e-mail address
  - Set a preferred language for the *SMARTUNIFIER Manager*.
- The role defines the permission of the user. It is mandatory to assign a role for the user. The following roles are available for use in the SMART**UNIFIER**.
  - **Administrator**: Full read and write access for the SMART**UNIFIER** Configuration and Administration.
  - **Reader**: Only read access for the SMART**UNIFIER** Configuration
  - **Writer**: Read and write access for the SMART**UNIFIER** Configuration
- Choose the account status: Active or Inactive.
  - **Active**: User account is activated and ready to use.
  - **Inactive**: User account is deactivated and cannot be used until it is activated again.
- Set an initial password for the first login of the new user.
- After all mandatory fields are filled in, click the "Save" button **(4)**.

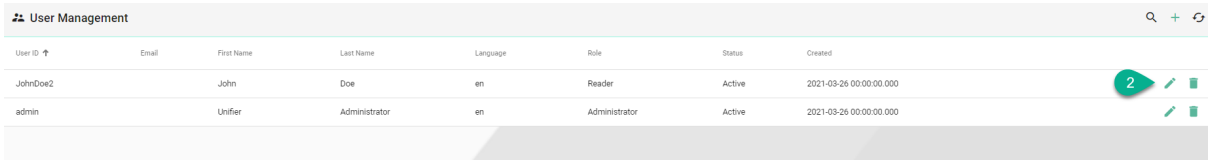


Edit a user

This procedure describes how to edit an existing user account.

- Select the SMART**UNIFIER** User Management perspective **(1)**.

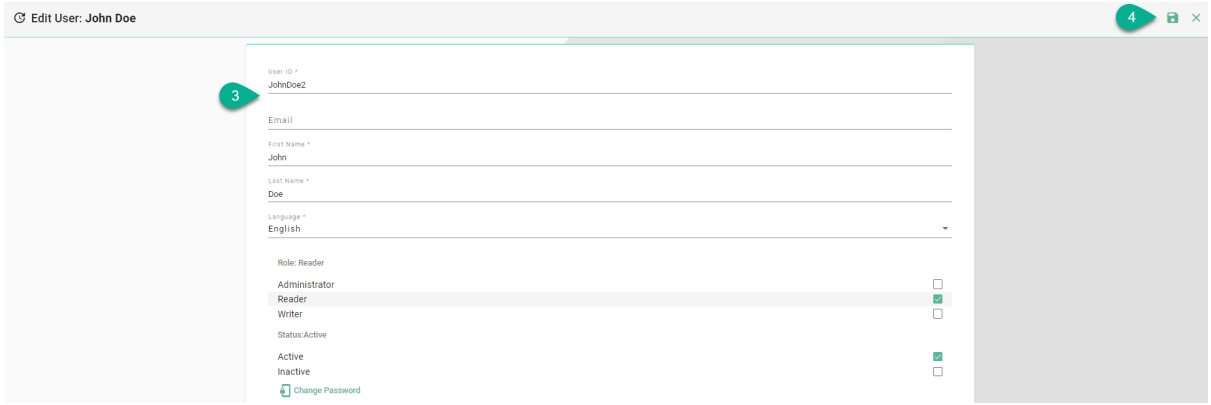
- Click the "Edit" button **(2)**.



In the "Edit" view the user account can be redefined **(3)**.

- update the user details: user id, first and last name, email address
- change the language
- edit the user permission: Administrator, Writer or Reader
- **activate** or **inactivate** the user account

- change the password



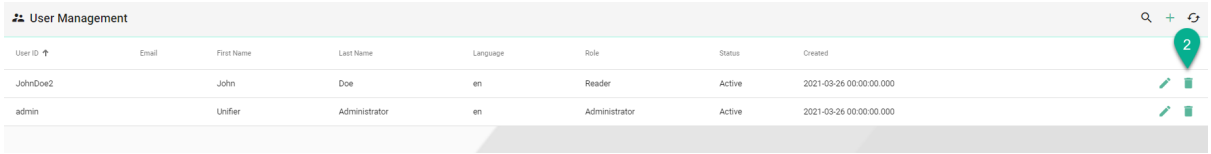
- After editing, click the "Save" button **(4)**.

Delete a user

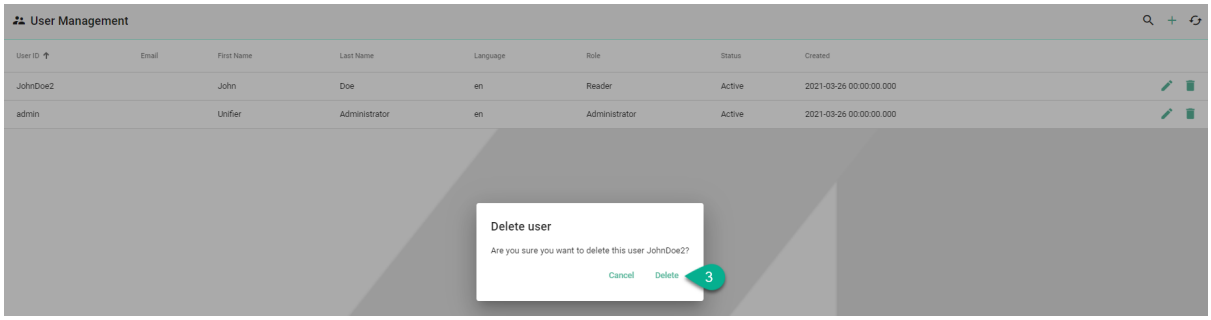
This procedure describes how to delete a user account.

- Select the SMART**UNIFIER** User Management perspective **(1)**.

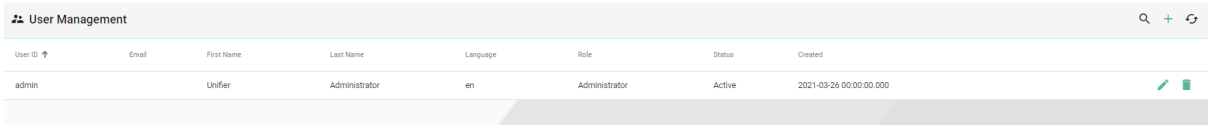
- Click the "Delete" button **(2)**.



Confirm by selecting the "Delete" button **(3)**.



The user account is deleted and no more visible in the SMART**UNIFIER** User Management perspective.



Monitoring

The Smart Unifier Manager offers a metrics API that allows monitoring of the current status of deployed instances and deployment endpoints. This API is accessible via a REST GET request

Deployments

http://localhost:9000/api/metrics/v1/deployments

Example


```
[
  {
    "id": "9a2bd6bb-8a3a-4e92-a125-06fc2f1d0ebf",
    "group": "demo.fab01.line4",
    "name": "LatheMachine_01",
    "version": "latest",
    "state": "Started",
    "communicationState": "Connected",
    "endpoint": {
      "id": "local",
      "group": "default",
      "name": "Default",
      "type": "Local",
      "state": "Running",
      "startupType": "Automatic"
    },
    "channels": [
      {
        "id": "1ea57580-70ae-4a25-ae61-3a47ba5b0c05",
        "group": "demo.fab01.line4",
        "name": "MachineContextInMemoryChannel",
        "version": "latest",
        "state": "Connected",
        "error": ""
```


(continues on next page)

(continued from previous page)


```
      },
      {
        "id": "39ed9664-dcef-45a6-bb38-13330645b039",
        "group": "demo.fab01.line4",
        "name": "MachinePartCounterGrpcClientChannel",
        "version": "latest",
        "state": "Connected",
        "error": ""
      },
      {
        "id": "7e4c7f49-6e4a-4aee-a093-5cc5a0bbe38c",
        "group": "demo.fab01.line4",
        "name": "MachineStatusGrpcClientChannel",
        "version": "latest",
        "state": "Connected",
        "error": ""
      },
      {
        "id": "b2bd8ad7-b549-4018-94b9-e345489a22de",
        "group": "demo.fab01.line4",
        "name": "MachineOpcUaClientChannel",
        "version": "latest",
        "state": "Connected",
        "error": ""
      }
    ]
  },
  {
    "id": "6edc8a83-e7a5-492b-8dc2-fbd0965331d5",
    "group": "demo.fab01.line4",
    "name": "LatheMachine_02",
    "version": "latest",
    "state": "Stopped",
    "communicationState": "Stopped",
    "endpoint": {
      "id": "local",
      "group": "default",
      "name": "Default",
      "type": "Local",
      "state": "Running",
      "startupType": "Automatic"
    },
    "channels": []
  }
]
```

Fields

Field	Description
`id`	UUID of the instance
`group`	The group of the instance
`name`	The name of the instance
`version`	Version of the instance
`state`	Current operational state of the instance (e.g., `Started`)
`communicationState`	Communication status (e.g., `Connected`, `Stopped`)
`endpoint`	Endpoint configuration object (see *Endpoint*)
`channels`	List of channel objects (see *Channels*)

Endpoint

Location where the instance is deployed with additional information about the endpoint

Field	Description
`id`	Identifier of the endpoint
`group`	Group the endpoint belongs to
`name`	Name of the endpoint
`type`	Type of endpoint (`Local`, `Agent`)
`state`	State of the endpoint
`startupType`	Startup mode (`Automatic`, `Manual` or `disabled`)

Channels

Each machine may contain one or more communication channels.

Field	Description
`id`	Unique identifier for the channel
`group`	The group or namespace the channel belongs to
`name`	The name of the channel
`version`	Version string, usually `"latest"`
`state`	State of the channel (e.g., `Connected`)
`message`	Any relevant message from the channel
`error`	Any error information if present

States

State	Description
`Stopped`	The channel is not running.
`Starting`	The channel is being started.
`Connecting`	Attempting to establish a connection.
`Disconnected`	The connection has been lost or not yet established.
`StartFailure`	The channel failed to connect (e.g. due to configuration issue or destination not available).
`RunningFailur`	The channel encountered an error while running.
`StopFailure`	An error occurred while trying to nicly stop the channel.
`Connected`	The channel is successfully connected and operational.
`Stopping`	The channel is in the process of shutting down.

Endpoints

hhttp://localhost:9000/api/metrics/v1/endpoints

Example


```
[
 {
   "id": "local",
   "group": "default",
   "name": "Default",
   "type": "Local",
   "state": "Running",
   "startupType": "Automatic"
 }
]
```

CHAPTER

SEVEN

GETTING HELP

Having trouble? We would like to help!

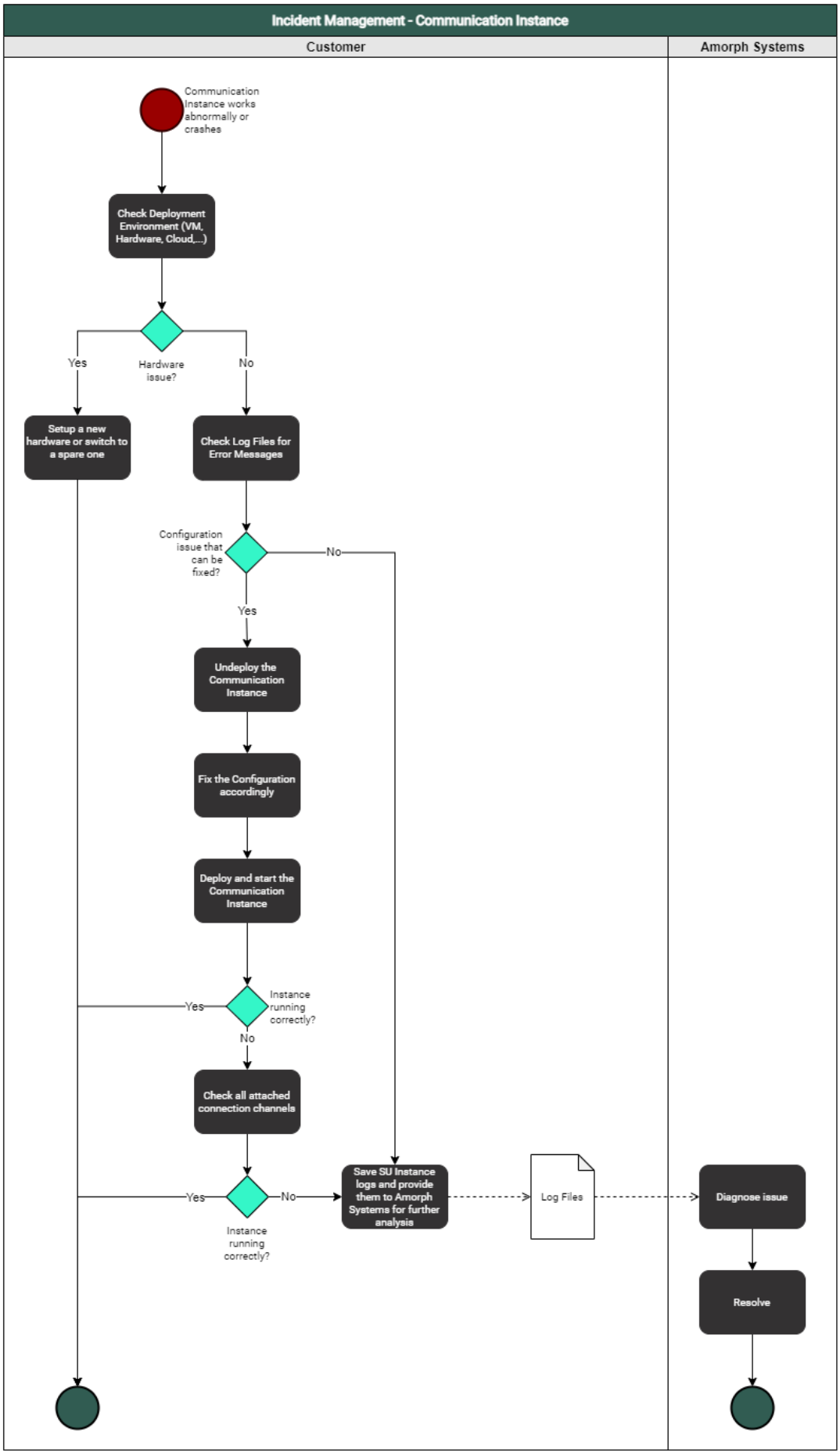
- In case of malfunctioning SU Instances check out the *Troubleshooting* section
- Try the *FAQ* - it's got answers to regularly asked questions
- Check out the *Glossary* if some terminology is not clear

Troubleshooting

Communication Instances

Determine if there is an issue with the deployment environment (VM, Cloud, other Hardware) where the Communication *Instance* is operated on.

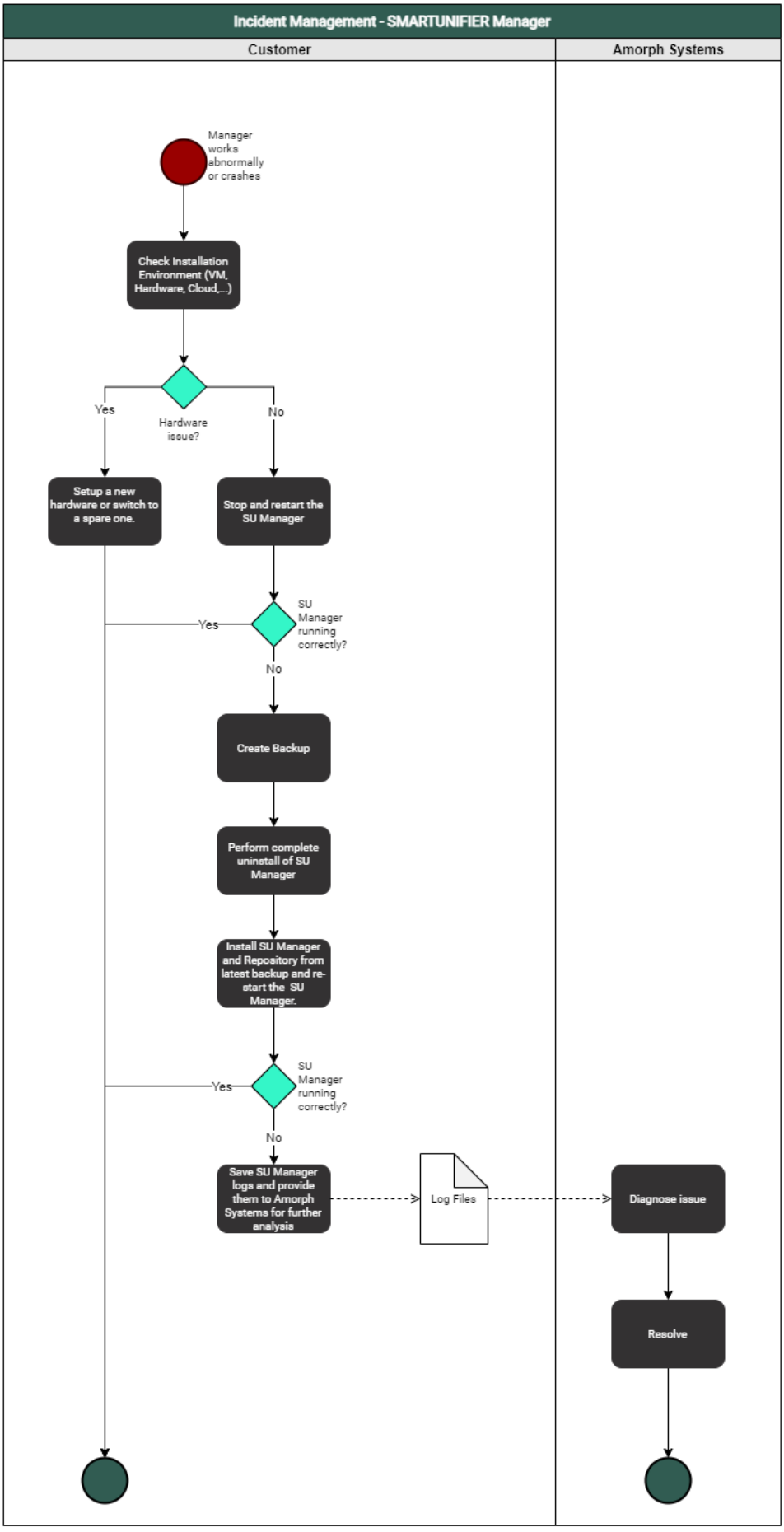
- In case of a HW problem setup a new HW (or switch to a spare HW). Ensure to place the correct security certificates on the new HW. Perform a new *deployment* of a new SU Instance with *SU Manager* on the new HW.
- In case, the HW is operating correctly navigate to the log file of the deployed instance **./SmartUnifierManager/deploy/<deployment-id>** and check for error messages.
  - If there is a configuration issue which can be fixed:
    * Undeploy the Communication Instance
    * Fix the configuration issue accordingly
    * *Deploy* and start the Communication Instance
  - If there is a configuration issue which can not be fixed save the log files and contact Amorph Systems through the Support Portal for further assistance


Incident Management - Communication Instance
Customer
Amorph Systems
Communication Instance works abnormally or crashes
Check Deployment Environment (VM, Hardware, Cloud,...)
Hardware issue?
Yes
No
Setup a new hardware or switch to a spare one
Check Log Files for Error Messages
Configuration issue that can be fixed?
No
Yes
Undeploy the Communication Instance
Fix the Configuration accordingly
Deploy and start the Communication Instance
Instance running correctly?
Yes
No
Check all attached connection channels
Instance running correctly?
Yes
No
Save SU Instance logs and provide them to Amorph Systems for further analysis
Log Files
Diagnose issue
Resolve

SMARTUNIFIER Manager

Determine if it is a HW problem on the HW where SMART**UNIFIER** Manager is operated.

- In case of a HW problem, setup a new HW or switch to a spare HW. Perform installation of SU Manager and Repository from latest backup and re-start the Manager on the new HW.
- In case HW is operating correctly stop and restart the Manager
- If the Manager is still not running correctly:
  - Create a Backup
  - Perform a complete uninstall of the Manager
  - Install the Manager with the Repository from the latest backup and start the Manager
- If the Manager is still not working navigate to **./SmartUnifierManager/log** and save the log files (debug.log and info.log) and contact Amorph Systems through the Support Portal for further assistance


Incident Management - SMARTUNIFIER Manager
Customer
Amorph Systems
Manager works abnormally or crashes
Check Installation Environment (VM, Hardware, Cloud,...)
Hardware issue?
Yes
No
Setup a new hardware or switch to a spare one.
Stop and restart the SU Manager
SU Manager running correctly?
Yes
No
Create Backup
Perform complete uninstall of SU Manager
Install SU Manager and Repository from latest backup and re-start the SU Manager.
SU Manager running correctly?
Yes
No
Save SU Manager logs and provide them to Amorph Systems for further analysis
Log Files
Diagnose issue
Resolve

FAQ

Does SMARTUNIFIER provide caching/buffering of data?

Yes, SMART**UNIFIER** is capable of supporting caching of messages using file buffer (Spool) for message transfer to external middleware like MQTT. This functionality can be provided as part of a SMART**UNIFIER** Communication Channel and dependent on the used communication protocol of the respective channel.

Is it possible to set different buffering options for different channels?

Yes, each communication channel of SMART**UNIFIER** can provide a different buffer size and further options.

Does SMARTUNIFIER enable data pre-processing, cleansing, filtering and optimization of data?

Yes, this is a core feature of SMART**UNIFIER**. SMART**UNIFIER** provides powerful capabilities for any kind data preprocessing, cleansing, filtering and optimization. The capabilities of SMART**UNIFIER** in this respect range from simple calculations, unit conversions, type conversions and reformatting up to arbitrary processing algorithms of any complexity.

Does SMARTUNIFIER enable data aggregation?

Yes, SMART**UNIFIER** enables data aggregation and reformatting with any level of complexity.

Does SMARTUNIFIER provide short term data historian features?

Yes, historic telemetric data (of variable time horizons; size limited by used HW) can be monitored by usage of SMART**UNIFIER**'s logs which can record all communication activities of a SMART**UNIFIER** Instance incl. telemetric data. SMART**UNIFIER**'s Log data can afterwards be forwarded by usage of a dedicated Communication Channel to any (and also multiple) upper-level monitoring or analytics system. Alternatively SMART**UNIFIER**'s Logs can be accessed directly by any external IT application (remote access to HW device is required).

Yes, SMART**UNIFIER** can create any number of OPC-UA Servers and/or Clients within just one Communication Instance.

Does SMARTUNIFIER support standard number of connections to OPC-UA Clients?

Yes, SMART**UNIFIER** supports a virtually unlimited number of client connections per OPC-UA Server. Physically the number of connections is limited by number of subscriptions per session, number of data objects and size per subscription as well as by HW and network constraints. SMART**UNIFIER** allows to operate multiple OPC-UA Servers and/or OPC-UA Clients within each single SMART**UNIFIER** instance for northbound and/or southbound communication.

Does SMARTUNIFIER support brokering to MQTT Server?

Yes, SMART**UNIFIER** supports any number of MQTT connections. One single SMART**UNIFIER** Instance can connect to one or multiple MQTT brokers (e.g., for different target systems) and is able to communicate bi-directional.

Which southbound protocols are offered with SMARTUNIFIER?

SMART**UNIFIER** supports many protocols like e.g.,

- Siemens S7, S7-2
- OPC-UA
- Beckhoff
- MQTT
- Modbus-TCP
- file-based (different formats like CSV, XML, JSON, any binary format)
- SQL

...and many more to come continuously. Specific protocols can be provided based on customer request. Therefore please contact Amorph Systems (www.amorphsys.com).

Does SMARTUNIFIER enable pre-aggregation of additional sensor data and/or more devices (rule based), for e.g., temperature monitoring?

Yes, SMART**UNIFIER** allows to connect any number of telemetric data sources to a SMART**UNIFIER** Instance. Rule-based pre-aggregation and pre-processing of additional sensor data is supported with any level of complexity. This ranges from simple pre aggregation/pre-processing up to complex utilization of advanced AI or ML algorithms.

Does SMARTUNIFIER support processing of active cloud commands? (e.g., System Manager AWS / AWS Agent)

Yes, SMART**UNIFIER** provides a RESTful API to execute Shell Commands (e.g., Start/Stop Instance, etc.). Thus, active cloud commands are supported. In addition, also commands from other external IT-Systems (e.g., MES, ERP, AWS Systems Manager etc.) are possible. Furthermore if required SMART**UNIFIER** can be fully executed and operated within Cloud Environments (e.g., within AWS Cloud).

Which northbound protocols are supported by SMARTUNIFIER?

SMART**UNIFIER** supports many northbound protocols, like e.g.,

- OPC-UA
- MQTT
- WebSphere
- HTTP / REST
- any file based protocol
- SQL/any database
- Splunk
- Vantiq

...and many more to come continuously. Specific protocols can be provided based on customer request. Therefore, please contact Amorph Systems (www.amorphsys.com).

Does SMARTUNIFIER support international naming standards (example: EUROMAP 77, PackML)?

Yes, SMART**UNIFIER** is specifically designed to strongly support the incorporation of international standards (e.g., EUROMAP 77, 82, 83, 84, AutomationML, PackML, DFQ, SEMI SECS/GEM etc.) as well as company standards, by offering the capability to be able to build up specific SMART**UNIFIER** Information Models complying with these standards and incorporating full data semantics. There will be a one-time effort to implement such a standard in SMART**UNIFIER** as a respective Information Model and afterwards this Standard can be used for any communication across the whole customer IT Infrastructure. Also this includes flexible mapping from legacy protocols to new standard protocol and vice versa.

Does SMARTUNIFIER offer the ability to integrate with other systems and applications through REST Server APIs and Web Services for Operational purpose?

Yes, SMART**UNIFIER** features a REST API for operational purpose (e.g., instance start/stop service, configuration etc.)

Does SMARTUNIFIER offer a way to realize a flexible, configurable dataflow?

Yes, SMART**UNIFIER** features a configurable and highly performant rule-based engine (SmartMappings) based on different northbound and/or southbound input sources for realizing any dataflow (workflow) that is required in industrial environments. This covers communication sequences for identification, processing start, processing execution, processing end, results data pro-vision as well as detailed process data provision. Also commands from any upper-level IT-System can be processed and further transmitted to the production equipment (e.g., recipe management, NC program transfer etc.) External data flow engines / visualization apps (e.g., Node-Red, Grafana) can be connected.

Does SMARTUNIFIER enable Central Software Management?

Yes, all Information Models, Mappings and Deployment Features can be managed centrally. Furthermore, SMART**UNIFIER** features an easy to use REST API for operational purpose (e.g., instance start/stop service, configuration etc.).

Does SMARTUNIFIER enable Container Deployment?

Yes, SMART**UNIFIER** operation and deployment is fully based on Container-Technology (Docker). SMART**UNIFIER** Manager and Instances can be operated and deployed inside Docker Containers to any End Point within the network running Docker environment.

Which Operating System SMARTUNIFIER is supporting?

SMART**UNIFIER** runs on Windows, Linux, Mac and other OS supporting Java RT and Docker.

Does SMARTUNIFIER support onPrem Edge-Analytics?

Yes, SMART**UNIFIER** can be connected to any Edge-Analytics System SMART**UNIFIER** Logs can provide detailed information about all communication activities. These log data can either be provided by a dedicated Communication Channel to any upper level Analytics System (in any required format) or can be made locally accessible to any agent running locally on the HW.

Does SMARTUNIFIER support DevOps CI/CD Pipeline for installations and update?

Yes, SMART**UNIFIER** supports remote installation/update of Software from SMART**UNIFIER** Manager via Docker Registry SMART**UNIFIER** Instances (running in Docker Containers) can be updated, monitored and controlled remotely. Docker registry is also accessible from external systems if required.

Does SMARTUNIFIER enable Software Scalability?

Yes, SMART**UNIFIER** can scale from connection of one single equipment/device to virtually any number of equipment/devices by means of its decentralized architecture.

Does SMARTUNIFIER support the architecture of distributed systems?

Yes, SMART**UNIFIER** itself is a fully distributed and scalable IT system. With this architecture SMART**UNIFIER** is able to collaborate in any small or large IT environment. SMART**UNIFIER** is open to reliably collaborate in large sites.

Does SMARTUNIFIER provide the ability to directly communicate with other Devices or IT-Systems through standard protocols and also supports Load-Balancing?

Yes, SMART**UNIFIER** can communicate with any other Devices or IT-Systems and also address load balancers for optimized feeding of data to any message brokers or data forwarder.

Does SMARTUNIFIER provide the ability for data to be ingested as a consolidated batch (File Transfer)?

Yes, SMART**UNIFIER** can use any file in any format as input source and also as output destination.

Does SMARTUNIFIER provide the ability to create custom connectors to ingest data from arbitrary sources?

Yes, the capability to be able to realize custom connectors for any data source is one of the core elements of SMART**UNIFIER**'s architecture.

Is SMARTUNIFIER able to push operational data to an Edge-Gateway?

Yes, SMART**UNIFIER** can receive operational data from any device or IT-System and push it to an Edge-GW. E.g., OPC-UA, MQTT and HTTP/REST are supported. Also, many other protocols can be used therefore.

Does SMARTUNIFIER provide Software Monitoring?

Yes, each SMART**UNIFIER** Instance creates detailed logs that document every communication activity. These logs can be made accessible to any external system e.g., by a dedicated monitoring communication channel. Moreover, SMART**UNIFIER** Manager comes with a built-in Monitoring Dashboard that allows monitoring of the distributed SMART**UNIFIER** Instances.

Does SMARTUNIFIER support Monitoring integration?

Yes, this is possible; Each SMART**UNIFIER** Instance creates detailed logs that document every communication activity. These logs can be made accessible to any external system e.g., by a dedicated monitoring communication channel. In addition, SMART**UNIFIER** is able to send any

kind of monitoring message (e.g., based on status changes or other events (e.g., time triggered) to any (or multiple) upper level monitoring system in any required format.

Does SMARTUNIFIER provide certificate handling?

Yes, SMART**UNIFIER** can handle certificates and establish state-of-the-art secured connections (e.g., TLS, secured MQTT, secured OPC-UA, etc.).

Is it possible with SMARTUNIFIER to limit access to data?

Yes, SMART**UNIFIER** Instances work on independent Windows/Linux computer units. Data may be stored temporarily on these HW devices as logs or for buffer (cache) purposes. This temporary data can be protected by assigning the HW with appropriate access rights and user roles.

Does SMARTUNIFIER support services for security supervision and security monitoring?

Yes, SMART**UNIFIER** creates detailed logs regarding all communication activities (and other activities) it performs. With SMART**UNIFIER** it is possible to integrate with any external security supervision/monitoring system (e.g., Splunk) and provide on-line log files (in any required format) to these systems by usage of a dedicated monitoring communication channel.

Does SMARTUNIFIER support End-to-End transport encryption (to Northbound and Southbound)?

Yes, SMART**UNIFIER** can support End-to-End transport encryption for southbound and northbound communication channels.

Does SMARTUNIFIER enforce secure individual authentication for all users?

Yes, SMART**UNIFIER** supports individual user authentication.

Does SMARTUNIFIER support Windows Active Directory (AD)?

Yes, SMART**UNIFIER** supports *Windows Active Directory*.

Does SMARTUNIFIER support a (configurable) secure remote access?

Yes, Secure remote access to SMART**UNIFIER** Manager and SMART**UNIFIER** Instances is possible by standard Windows or Linux tools (e.g., SSH).

Can SMARTUNIFIER protect unsecured Shop Floor devices from office network through isolation?

Yes, a SMART**UNIFIER** Instance can be deployed locally near an equipment/device and map any unsecured equipment/device interface into a secured protocol (e.g., OPC-UA, MQTT). This way "unsecured data streams" coming from an equipment/device can be transferred to any northbound system in a secured way (isolation of the equipment/device). The same principle can be also applied when sending control parameters (e.g., screwer params, NC programs, recipes, ...) or commands from a northbound system to the equipment/device.

Does SMARTUNIFIER support malware protection concepts (e.g., support of standard Anti-Virus Software)?

Yes, SMART**UNIFIER** works with any standard malware protection software incl. McAffee, NOD and many others.

Is SMARTUNIFIER secure by design (e.g., secure coding guidelines, use of open source code, pentesting)?

SMART**UNIFIER** was developed according to state-of-the-art coding principles and on request we are willing to let perform any checks, verifications, pen testing as required to validate the software. Especially for realizing communication channels and implementing protocols, state-of-the-art Open Source Libraries are used and constantly updated to the newest versions available.

Does SMARTUNIFIER support a range of transmission/infrastructure protocols (e.g., IPV4/IPv6)?

Yes, with SMART**UNIFIER** (depending on used HW) IP4/IP6 are supported.

- LAN: Up to 4x Gbit Ethernet Intel i211
- Wireless LAN: 802.11ac dual antenna + BT 4.2
- Cellular communication: LTE/WCDMA/GSM/GNSS

USB: Up to 8 ports, 2x USB 3.0, Up to 6x USB 2.0

- RS232 serial port

Also other transmission/infrastructure protocols can be supported on request but may require additional HW.

Does SMARTUNIFIER provide the ability to handle intermittent connectivity of sources (data/event redelivery and failure modes)?

Yes, intermittent connectivity of sources can be handled by SMART**UNIFIER** Communication Channels. Based on rules, data/event redelivery can take place, failure modes can be activated, and escalation procedures to northbound systems can be triggered.

Does SMARTUNIFIER reduce unnecessary traffic on shop floor network to protect device interfaces from traffic overload?

Yes, a SMART**UNIFIER** instance can be deployed locally nearby the equipment on any suitable HW device. The SMART**UNIFIER** instance can then be configured to communicate to the connected southbound equipment/devices by using a separate physical network port and this way isolate the device from unnecessary traffic coming from the northbound network.

Does SMARTUNIFIER support low Latency between Southbound and Northbound Interfaces?

Yes, SMART**UNIFIER** provides high performance / low latency by its distributed architecture consisting out of small SMART**UNIFIER** Instances (i.e., no central bottlenecks like e.g., a middleware broker/database). Furthermore, SMART**UNIFIER** features an integrated compiler that creates native Bytecode for the interfaces to be executed within the SMART**UNIFIER** Instances. This makes the SMART**UNIFIER** highly performant, since no slow scripting language nor any slow interpreter is used to provide the connectivity functionality.

Is it possible with SMARTUNIFIER to ensure a consistent setting of time stamps for events (NTP)?

Yes, this is possible.

Is it possible to use UNICODE for operational data?

Yes, it is possible to use UNICODE with SMART**UNIFIER** (e.g., for OPC-UA).

Is stability of SMARTUNIFIER s API given? Is the API stable across releases?

Yes, SMART**UNIFIER** is a standard product from Amorph Systems. Interface stability is given and stable across new product releases. Furthermore, interfaces are versioned and under controlled release management (i.e., different versions of interface, Information, Models and Mappings can be maintained and deployed in a controlled mode).

Which tools for development, deployment and error analysis can be used with SMARTUNIFIER ?

For extension, deployment and error analysis of SMART**UNIFIER** (e.g., development of new Information Models, pre-processing, aggregation etc.) widely-used and accepted state-of-the-art development environments and powerful standard tools may be used, e.g., Eclipse, Maven/sbt, Jenkins, Docker. For Error Analyses detailed logs created by SMART**UNIFIER** can be used and analysed with any analytics tools.

What is the cost model of SMARTUNIFIER ?

Please refer to Amorph Systems (www.amorphsys.com) for prices for SMART**UNIFIER** . In general, the following policies apply:

- SMART**UNIFIER** Manager is free of charge
- For SMART**UNIFIER** Instances a yearly license fee is charged

Does Amorph Systems offer reliable support for SMARTUNIFIER ?

For many years, Amorph Systems is providing first class support and intensive care to all of its customers. This covers all products and solutions that were delivered and operated in Industrial Areas as well as in Air Traffic Industry. For customer references please refer to Amorph Systems (www.amorphsys.com).

What support levels (SLAs) are supported?

Different levels of services (8x5, 8x7 up to 24x7) are available upon request from Amorph Systems (www.amorphsys.com).

Does SMARTUNIFIER support multiple languages?

Yes, SMART**UNIFIER** can support multiple languages. Currently the GUI is available in English and German language. In case more languages are required, please contact Amorph Systems (www.amorphsys.com)

Does Amorph Systems provide relevant training capabilities for operating SMARTUNIFIER and for engineering of Information Models and Mappings?

Yes, SMART**UNIFIER** is a simple to use standard product and was specifically designed as a powerful tool to enable the end customers themselves to provide seamless equipment/device as well as IT-Systems interconnectivity within their industrial environments.

Therefore, Amorph Systems trains customers to configure, deploy and operate SMART**UNIFIER** in their environments. Moreover, we can give advanced trainings, so that the customers can also implement new interfaces, new channels, new, Information Models and new Mappings on their own.

Glossary

Arrays
: An Array (as an Information Model Node Type) is a container object that holds a fixed number of values of a single type.

Configuration Components
: Configuration Components are Information Models, Communication Channels, Mappings, Device Type and Communication Instances, used to realize an integration scenario.

Commands
: Commands (as an Information Model Node Type) are functions like the methods of a class in object-oriented programming. The scope of a Command is bounded to the Information Model it belongs.

Communication Channels
: Communication Channels or simply Channels, refer to a transmission medium. A Channel is used to convey information from one or several senders (or transmitters). Communicating data from one location to another requires some form of pathway or medium. These pathways are called Communication Channels, and the information is transmitted with the help of communication protocols. Each Information Model has one or many Channels, and each Information Model can choose which Channel it subscribes to. The information is transmitted through the Communication Channels in both directions: from the external system to the SMART**UNIFIER** application and vice versa.

Custom Types
: Custom Data types are defined by the user for a Node Type.

Data Types
: Each Node Type has a Data Type. Data Types can be either a Simple Type or a Custom Type - depending on the Node Type.

Deployments
: With the SMART**UNIFIER** Deployment capability Instances can be deployed to any IT resource (e.g., Equipment PC, Server, Cloud) suitable to execute a SMART**UNIFIER** Instance.

Deployment Endpoints
: Deployment Endpoints are used to identify the location of a Deployment (e.g., AWS Fargate, Docker)

Device Types
: Device Type contains one or multiple Mappings. Each Mapping contains one or multiple Information Models and its associated Communication Channel. Within a SMART**UNIFIER** Device Type it is possible to overwrite existing Communication Channel

configurations. Device Types are especially important, when having to connect several similar equipment or devices that share the same communication parameters. In these cases it is sufficient to define only one Device Type and the settings of this Device Type can be reused across all Instances.

Events
Events (as an Information Model Node Type) represent an action or occurrence recognized by SMART**UNIFIER**, often originating asynchronously from an external data source (e.g., equipment, device). Computer events can be generated or triggered by external IT systems (e.g., received via a Communication Channel), by the SMART**UNIFIER** itself (e.g., timer event) or in other ways (e.g., time triggered event).

File Consumer
This Communication Channels offers the capability to read-in files (e.g., CSV, XML, and JSON). The File Consumer monitors an input folder that is specified in the configuration.

File Tailer
This Communication Channels offers the capability to read-in files (e.g., CSV, XML, and JSON). The File Tailer monitors a specific file, which is specified in the configuration.

InfluxDB
This Channel offers connectivity to an InfluxDB. InfluxDB is an open-source time series database.

Information Models
Information Model describes the communication related data, which is available for a device or IT system. Each device or IT system is represented by an Information Model.

Instances
An Instance represents an application that handles the connectivity. Instances can be deployed to any suitable IT resource (e.g., Equipment PC, Server, Cloud), and provide the connectivity functionality configured. Therefore, a SMART**UNIFIER** Instance uses one or multiple Mappings and selected Communication Channels from a previously defined Device Type.

Lists
A List (as an Information Model Node Type) representing collections of Node Types (e.g., Variables, Properties, Arrays, and other Lists).

Mappings
Mapping represents the SMART**UNIFIER** component that is defining when and how to exchange/transform data between two or multiple Information Models. In other words it is acting as a translator between the different Information Models. One Mapping consists of one or multiple Rules.

MQTT
This Communication Channel offers the capability to send data using the messaging protocol MQTT. MQTT is a lightweight publish/subscribe messaging transport for connecting remote devices with minimal network bandwidth.

Node Types
Node Types are elements within an Information Model. Node Types are Variables, Properties, Events, Commands and also collections such as Arrays and Lists. Each Node Type has a Data Type that defines if the Node Type is a predefined Data Type or a custom Data Type.

OPC-UA
Is a machine to machine communication protocol for industrial automation.

Predefined Types
Predefined Data Types (e.g., String, Integer, Double, etc.) are available for the definition types - Variables, Properties, Arrays, Lists (e.g., String, Integer, Boolean).

Properties
Properties (as an Information Model Node Type) are used to represent XML attributes.

REST Client
This Communication Channels offers the capability to consume and operate with resources exposed by REST Servers.

REST Server
This Communication Channels offers the capability to provide resources employing the HTTP communication protocol.

Rules
A Rule contains a Trigger that defines when the exchange/transformation takes place and a list of actions that are defining how the exchange/transformation is done.

Manager
The Web application SMART**UNIFIER** Manager is used to create and monitor SMART**UNIFIER** Instances.

Source
In the Mapping sources are Node Types that are mapped to targets.

SQL DB
This Communication Channel offers the capability to connect to a SQL Databases (e.g., MariaDB, SQLServer, PostgreSQL, ORACLE, HSQLDB, and DB2).

Target
In the Mapping targets are Node Types that receive data assigned from a source.

Trigger
The Trigger defines when the exchange/transformation data between two or multiple Information Models takes place.

User Management
User Management allows the administrator to create users accounts, to assign permissions as well as to activate or to deactivate the user accounts.

Variables
Variables in an Information Model represent data values and structures.

What has changed in 1.10.x

1.10.1

Changes

- Improved performance by caching channel configurations

Bug fixes

- Deployment endpoints not reinitialization after restoring a backup

- Device type cannot be saved after removing channel
- Agent state not updated after renaming
- Environment variable short not opening anymore the developer tools on firefox

Added

- OPCUA Client: Support for events
- Implicit type conversion for multi rules

1.10.0

Changes

- Update from Java 11 to Java 21
- Update to Angular 19
- Timestamps in channel state changes are now using UTC time zone
- Several small ui updates and improvements
- Improved logging in Mapping type conversion helpers
- Environment variables are now identified by a UUID

Added

- Mapping: Seq() can now be assigned to list elements in models
- Metrics API for instance states
- Introduced startup types for instances and endpoints
- Introduced new MQTT Client channel with more flexible configuration options and support of wildcards

What has changed in 1.9.x

> **Hint**
>
> Deployment options like AWS, Docker, and SSH are removed from the standard product and available upon request. Please contact Amorph Systems for more details.

1.9.12

Changes

- Windows Service: Java home path now read from application.conf instead of JAVA_HOME environment variable

Added

- InMemory: Added support for initializing simple arrays

1.9.11

Added

- Introduced new MQTT Client channel with more flexible configuration options and support of wildcards

1.9.10

Bug fixes

- Deployment endpoints not reinitialization after restoring a backup
- Fix deployment endpoints states for agents not updated correctly

1.9.9

Bug fixes

- Mapping: FixedRateScheduler cannot be started manually
- Manager: Deployment endpoint state was not updated after restore of backup
- OPCUA Client: Connection state not updated correctly after reconnect

Changes

- Disabled lock file for service installation of Manager and Agent
- OPCUA Client: Ignore invalid BuildInfoType from OPC-UA Servers
- Updated procrun to 1.4.1 for manager and agent service installation

1.9.8

Bug fixes

- Rest client config is not shown when the name of the variable is BODY,HEADER,PARAMETERS or STATUSCODE

Added

- InfluxDB v1: Added support for reading from influx database

1.9.7

Bug Fixes

- Sql Database: Fix database connection not cleaned up after disconnect

1.9.6

Bug Fixes

- Updated dependencies to fix version conflict in used libraries

1.9.5

Changed

- Sql Database: More granular connections settings for poolsize, timout, ...

1.9.4

Added

- Rest Client: Added support for additional OAUTH2 type
- Rest Server: Server can now use certificates stored in the Windows certificate store

Changed

- Endpoint don't needs to be stopped before doing restore

Improvements

- Backups can be downloaded as zip files

1.9.3

Added

- EMail: Added option to use outgoing server without any security

Changed

- Max logging to 180 days

1.9.2

Added

- Saving mappings without compiling

Bug Fixes

- Rest Client: Fixed configuration for selecting HTTP-Version

1.9.1

Added

- Rest Client: Support for OUTH2.0 added
- Rest Client: Support for body type "text" added
- Rest Server: Support for body type "text" added

Improvements

- Possibility to select which entries of the database to backup / restore.
- Added confirmation prompts for artifact deletion and reindexing.
- Visual distinction of complex and simple variables in Model editor / view
- Documentation updates: Expanded guides for backup, Oracle drivers, and more.
- OPC-UA client: Improved logging
- MQTT layers: Improved logging

Bug Fixes

- Restoring of deployments not working
- Incorrect state of an agent deployment endpoint shown in gui
- Creation of environment variables not working
- OPC UA Client: Fixed subscription group "None" not recognized

1.9.0

Added

- Validation for the SMART**UNIFIER** configuration component.
- Environment variables to store common configurations across multiple Communication Channels.
- New mapping trigger type "Timeout Scheduler".
- Implicit data type conversion within mappings via drag-and-drop.
- Mapping helpers such as *equals*, *formatDateTime*, *parseDateTime*, *mapAndAssignChildren* for easier rule creation.

Improvements

- Introduced "Communication Status" to indicate the status of communication between channels.
- Updated MQTT Communication Channel to allow subscriptions on complex variables.
- Updated SMART**UNIFIER** Manager Interface framework to Angular 16.
- Improved logging in Communication Channels to make it easier to react to certain events.
- Multiple improvements to enhance the stability of the SMART**UNIFIER** Manager and Communication Instances.

Bug Fixes

- Reconnect issue in InfluxDB Communication Channel after connection loss.
- Connection state issue in OPC UA Client Communication Channel.
- Absence of notification when the Information Model compile fails.

What has changed in 1.8.x

Added

- Added: Deployment Agent that allows to deploy and monitor SMART**UNIFIER** Communication Instances remotely.
- Added: Downloading feature to get the logs of an SMART**UNIFIER** Communication Instance via the Deployment view.
- Added: Alert feature, enabling users to monitor running Communication Instances and create customizable alerts to receive notifications in case of errors.

- Added: Validation feature has been added to check the validity of configuration components such as Information Models, Communication Channels, Mappings, Device Types, and Communication Instances, providing assurance that the they are free from faults and errors.
- Added: New Communication Channels (Email, InfluxDB V2).

Improvements

- The Credential Manager now has the ability to manage tokens in addition to other credentials.
- To ensure all Communication Instances are properly started after a backup, an automatic restart feature has been implemented for Instances that were started prior to the backup.
- A more robust logging framework has been introduced, resulting in improved performance across the system.
- Visually enhanced dashboard charts for CPU usage, RAM usage, and messages per second to provide an improved overall appearance.

Bug Fixes

- Fixed: Data type handling in the Information Model JSON-Import plugin.
- Fixed: Issue affecting the WebSocket channel's reconnection behavior has been fixed.
- Fixed: Enabled editing of action timeout in SecsGem Channel.
- Fixed: Arrow key logic in Information Model view.

What has changed in 1.7.x

1.7.0

Added

- Added: OPC-UA model import - provides the possibility to generate an OpcUa Information Model using a XML-file or connecting to the OpcUa server.

Improvements

- Introduced actions with conditions within the Mapping Rule configuration.
- Added option for multi trigger Rule within the Mapping configuration. Simple one to one Mapping of variables.
- Added option to enable consumer/producer for the variable node within the MQTT Communication Channel configuration.
- Extended local deployment to flexibly choose the location of the Instance deployment on your system by adding local endpoints.
- Introduced Deployment states within the Deployment Instance operations.
- Improved Modbus implementation.
- Added tracing support for SECS implementation.

Bug Fixes

- Fixed: Variables under event duplicated when dragging in an action from the tree within Mapping.
- Fixed: Deploying of an Instance parsed the Mapping code.
- Fixed: Error while creating AWS Deployment Endpoint.
- Fixed: Log and status of Instances not working on SSH Deployments.
- Fixed: Temp folder is not cleaned up after restoring a backup.
- Fixed: REST client authentication issue.
- Fixed: Notification issues.

What has changed in 1.6.x

1.6.0

Added

- Added: New trigger types for Rules within the Mapping - Fixed Rate Scheduler and Fixed Delay Scheduler.
- Added: Simple Mapping option for data structures of the same type.
- Added: SSH Deployment that allows to deploy Communication Instances in different networks.
- Added: Notifications allow another way of monitoring deployed and running Communication Instances states.
- Added: The new SiteWise model export extension makes an integration with AWS IoT SiteWise more easy.
- Added: The new JSON model import extension that allows you to create Information Models automatically based on a given JSON structure.

Improvements

- Introduced configuration field within the Communication Channel configuration to define deviating or unsupported Information Model node naming (e.g., avoid naming conflicts with Scala keywords).
- Added option *TailFromEnd* to the File Tailer Communication Channel.
- Implemented List support for Command Replies.
- You can backup the internal database of the SMART**UNIFIER** Manager.
- Improved Dashboard for the monitoring of deployed and running Instances.
- Changed AWS account authentication from profile file to input fields for Access key id and Secret access key.

Bug Fixes

- Fixed: Deletion of Information Model custom nodes.
- Fixed: Type conversion issues with Milo unsigned types in the OPCUA Communication Channel.
- Fixed: Deadlock issue in the SECS/GEM Communication Channel.

- Fixed: Communication Instances show now the correct state when using a backup of a repository.
- Fixed: UI issue during the offline activation when entering a new license.

What has changed in 1.5.x

Added

- Mappings: Introduced rule scheduler as another trigger option.
- Communication Instance: Added the option to choose the version of the framework which the Communication Instance is using to allows backwards compatibility.
- Deployments: Added bulk actions to deploy, start, stop and delete multiple selected Deployments.
- Deployments: Added VM Arguments to configure the Java Virtual Machine (JVM) for the Communication Instance.
- Deployments - Docker: Introduced possibility to attach volumes to Docker containers.
- Administration - Logging Configuration: Allows to generate customized log4j configurations that can be used when deploying Communication Instances.

Improvements

- Administration - Restore: Introduced progress bar with log viewer for monitoring the restoring process of the repository.
- Communication Channels - REST Client: Introduced option to use parameters within the URL.
- Communication Channels - InfluxDB: Introduced support for Arrays.

Bug Fixes

- Information Models: Children nodes can have now the same name as their parent nodes.
- Administration - SCM (Gitea): Fixed issue of failing API requests when organizations are missing within Gitea.

What has changed in 1.4.x

1.4.0

Added

- Deployments: Introduced log viewer to show logs of deployed SMART**UNIFIER** Communication Instances.
- Deployments: Introduced flag "protected" for deployed SMART**UNIFIER** Communication Instances - requires a password to apply state changes on deployed Instances.
- Deployment Endpoint - Docker: Added TLS to protect the Docker daemon socket.

Improvements

- Manager: Introduced local Git repository to version SMART**UNIFIER** Communication Instances by default. The use of an Gitea repository can be configured optionally.

Bug Fixes

- Communication Channels - SECS: Not reconnecting when first attempt fails.
- Communication Channels - MQTT: Layer connection state not set correctly.
- Manager UI: Fixed Dashboard refreshing issue when Communication Instance changes from Stopped to Started.

What has changed in 1.3.x

1.3.0

Added

- Manager UI: Introduced backup and restore functionality for the SMART**UNIFIER** repository.
- Manager UI: Introduced Docker Java Image Manager to support the administration of Docker images for containerized deployments of Communication Instances.
- Manager UI: Introduced clone functionality for configuration components that allows to easily reuse components like Information Models, Communication Channels, Mappings, Device Types, Communication Instances as well as Deployment Endpoints.
- Manager UI: "About SMART**UNIFIER**" Pop-up
- Deployments: Added option to encrypt Communication Instances so that configuration files with sensitive data are secured.
- User Administration: Added support for Windows Active Directory (AD).
- Demonstrator: Added demo guide for high-availability (HA) use cases using a load balancer before two or more SMART**UNIFIER** Communication Instances.

Improvements

- Manager UI: Introduced additional "save & close" button for configuration pages to exit the configuration faster.

Bug Fixes

- Communication Channels: Fixed configuration issues on following Communication Channels: REST Client, MQTT and File Reader (JSON, XML, and CSV).
- Communication Channel - SQL Database: Added support for multiple nested lists inside an Information Model.
- Manager UI (macOS): Fixed deployed and started Communication Instance state - After restarting the SMART**UNIFIER** Manager the running Communication Instance was not displayed as running.

What has changed in 1.2.x

1.2.0

Important

Breaking Change: This release contains a major update of the SMART**UNIFIER** Framework. Instances configured in an older release will not work with this version. Please contact Amorph Systems for guidance on how to migrate SMART**UNIFIER** Instances from previous releases.

Improvements

- Improved architecture performance and stability by updating the framework to Scala version 2.13 and Java version 11.
- Communication Channels: Improved configuration of Communication Channels by enhancing the internal process of how the configuration forms are generated.
- Manager UI: Introduced new icons for several menu entries (Information Model, Mappings, Device Type, Instance, Deployment, Deploy and Undeploy) to improve usability.

Bug Fixes

- Manager UI: Fixed small UI styling issues.
- Communication Channel - IsoOnTCPClient: Fixed configuration issue.
- Mapping: Added check to make sure that the Rule name is valid.

What has changed in 1.1.x

1.1.6

Added

- Security Improvement: Encryption of credentials.
- Communication Channel: Simplified Communication Channels configuration using preconfigured Channel Types.

Bug Fixes

- Information Model: Removed Simple Type as data type for Events and Commands.
- Information Model: Ensure that "Save" button is only enabled when all mandatory fields are filled.
- Communication Channel: OPC-UA Client configuration issues.
- Communication Channel: File Consumer file handling when error occurs in the communication.
- Communication Channel: MQTT layer reconnects when no disconnect buffer is used.
- Device Type: Enable the configuration of the Communication Channels before saving the new Device Type.
- Instance: Fixed Instance starting issue.

1.1.5

Added

- Import/Export functionality for SMART**UNIFIER** Artifacts (Information Models, Communication Channels, Mappings, Device Types, Instances, Deployment Endpoints, Deployments) to allow transfer of artifacts in a simplified way.
- Communication Channel: Source code editor that displays the Channel configuration in JSON format.
- Security: Encryption of databases.

Improvements

- Communication Channel - SQL Databases: Handling of infinity values.

Bug Fixes

- Deployment: Fixed configurable refresh on deployment page.

1.1.4

Added

- Mapping: Option to enable/disable rules.
- Communication Channel - InfluxDB: Added Tags and allowing the renaming of variables via configuration.
- Communication Channel - SECS: Added SECS as a new Communication Channel type.
- Deployment - AWS: Introduced deployment of Instances on AWS using Fargate.

Improvements

- Manager UI: Updated to Angular version 11 to improve performance and usability.

Bug Fixes

- Framework: Paths that are containing the SMART**UNIFIER** Manager are now allowed to have spaces in it.
- Information Model: Fixed Copy and Paste issues of Nodes in the model editor.
- Communication Channel - SQL Database: Supports now all kinds of connection paths.
- Device Type: Alphabetically sorted list of Mappings.

1.1.3

Added

- Mapping: Introduced check that shows if a Rule is valid or invalid.
- Deployment: Added more default logging settings (Info, Debug, Trace, Warning).
- Deployment - Docker: Introduced health check for containers in order to determine whether the resource is operating normally.
- Repository: Added option to reindex (update) all implementations that are stored in the repository.

Improvements

- Communication Channel - MariaDB: Updated driver to version 2.6.2.

Bug Fixes

- Information Model: Fixed renaming of complex variable MemberType to make sure all dependent nodes are updated to the new name.
- Communication Channel: Fixed duplication of Communication Channels when clicking on "save".
- Device Type: Fixed several UI issues when clicking on "apply" and "save".

1.1.2

Added

- Manager UI: Added group filter for all artifacts.
- Communication Channel - MQTT: Buffering of messages when MQTT Client is not connected.

Improvements

- Communication Channel - SQL Database: Simplified configuration.

Bug Fixes

- Communication Channel - MQTT: Added ID to the MQTT persistence folder to avoid multiple clients conflicts.
- Instance: Prohibit Instance from stopping in case another Thread is running in a Channel Implementation.

1.1.1

Added

- Added Port and IP of the SMART**UNIFIER** Manager to the application.conf file.

Improvements

- Communication Channel: Introduce default configuration for Channels.

Bug Fixes

- Instance: Ensure that multiple used Communication Channels have only one configuration.
- Device Type: Fixed long loading time when accessing Rules in the Mapping.

1.1.0

Added

- Mapping: Allow the synchronous execution of commands.

Improvements

- Communication Channel - File Consumer: Improved logging and formatting of parsed files.

Bug Fixes

- Information Model: Fixed icon of model node "Array".
- Communication Channel - SQL Database: Fixed reconnect when connection to the database is lost.

What has changed in 1.0.x

1.0.1

Added

- Instance: Added schema validation for configuration values in Instances.

Improvements

- Manager UI: Updated to Angular 9 - making use of several performance increasing features.
- Communication Channel - OPCUA: Allow configuration of subscription attributes.

Bug Fixes

- Communication Channel: Removed null values from the configuration.

1.0.0

Added

- Added Device Type feature. Enables the possibility to group similar integration scenarios and the creation of multiple Instances based on a Device Type.
- Deployment - Docker: Added Docker deployment of Instances.

Improvements

- Communication Channel: Sort list of Channel Types alphabetically.

Bug Fixes

- Information Model: Removed simple data types when creating a new Event/Command.
- Mapping: Introduced check that evaluates if a Rule name already exists.